



Unidad Académica Cochabamba

PROGRAMACIÓN

CONCEPTOS Y EJERCICIOS

Lic. Danitza Eliana Solar Llanos
Lic. Lizbeth Jaramillo Martinez
Lic. Ana Marlene Ticona Flores

COCHABAMBA - 2020

TABLA DE CONTENIDOS

I. ALGORITMOS Y PROGRAMACIÓN	4
1.1 Evolución de las ciencias de la computación	4
1.1.1. La primera generación de computadoras (1938 – 1952)	6
1.1.2. La segunda generación de computadoras (1953 – 1962)	7
1.1.3. La tercera generación (1964 - 1971)	9
1.1.4. La cuarta generación (1971-1988)	9
1.1.5. La quinta Generación (1983-20...)	10
1.2 Componentes físicos y lógicos de un computador	11
1.2.1 Componentes físicos	11
1.2.2 Componentes lógicos del computador	13
1.3 Modelado de datos en binario de caracteres y números	15
1.3.1. Código BCD	15
1.3.2. Código ASCII	16
1.4 Conversión de sistemas de numeración	17
1.4.1. Sistema decimal	17
1.4.2. Sistema binario	17
1.4.3. Sistema octal	18
1.4.4. Sistema hexadecimal	18
1.4.5. Unidades de Almacenamiento	19
1.5 Definición de algoritmos	19
1.6 Características de los Algoritmos.	19
1.7 Lenguajes de programación para la representación de algoritmos	20
1.8 Ejercicios Propuestos	20
II. ESTRUCTURAS SECUENCIALES.	21
2.1 Metodología de resolución de problemas	21
2.2 Tipos de datos numéricos, lógicos y alfanuméricos.	21

2.3	Variables y Constantes	23
2.4	Expresiones aritméticas, lógicas y relacionales	23
2.5	Asignaciones	29
2.6	Entrada/Salida de Datos	29
2.7	Representación en Pseudocódigo y diagrama de flujo	29
2.8	Concepto de Secuencia	34
2.9	Ejercicios Propuestos	37
III.	ESTRUCTURAS SELECTIVAS.	38
3.1	Estructura de un Algoritmo de Selección	38
3.2	Estructura de selección Simple:	38
3.3	Estructura de selección Doble:	40
3.4	Estructuras de selección Compuestas	41
3.5	Estructura de selección Múltiple	42
3.6	Estructuras selectivas anidadas	43
3.7	Prueba de escritorio de la solución y depuración	45
3.8	Aplicaciones al algebra y cálculo	46
3.9	Ejercicios Propuestos	49
IV.	ALGORITMOS REPETITIVOS	50
4.1	Concepto de Bucle e iteración	50
4.2	Diseño de la Estructura de algoritmos repetitivos	50
4.3	Contadores y Acumuladores	50
4.3.1.	Contadores	50
4.3.2	Acumulador	51
4.4	Estructura Para (for)	51
4.5	Estructura Mientras – hacer (While)	52

4.6	Estructura Repetir – Hasta (Do - While)	53
4.7	Estructuras repetitivas anidadas	54
4.8	Aplicaciones al algebra y cálculo	55
4.9	Ejercicios Propuestos	58
V.	FUNDAMENTOS DE PROGRAMACION.	59
5.1	Proceso de codificación, compilación y ejecución de un programa	59
5.2	Estructura de un programa	61
5.3	Funciones proporcionadas por el compilador	63
5.4	Aplicaciones empleando estructuras secuenciales, selectivas y repetitivas.	64
5.5	Ejercicios Propuestos	65
VI.	ARREGLOS	67
6.1	Arreglos Unidimensionales	67
6.2	Ordenación y Búsqueda	73
6.2.1	Ordenación	73
6.2.2	Búsqueda	74
6.3	Arreglos Bidimensionales	75
6.4	Ejercicios Propuestos	79
VII.	SUB PROGRAMAS	81
7.1	PROCEDIMIENTO	81
7.2	Funciones	82
7.3	APLICACIONES CON FUNCIONES Y PROCEDIMIENTOS	83
7.4	Ejercicios Propuestos	85
VIII.	Bibliografía	86

I. ALGORITMOS Y PROGRAMACIÓN

Introducción

La computadora no solamente es una máquina que puede realizar procesos para darnos resultados, sin que tengamos la noción exacta de las operaciones que realiza para llegar a esos resultados. Con la computadora además de lo anterior también podemos diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aún, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos.

El diseño de soluciones a la medida de nuestros problemas requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones.

A las soluciones creadas por computadora se les conoce como **programas** y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos. Lo anterior nos lleva al razonamiento de que un **programa** nos sirve para solucionar un problema específico.

1.1 Evolución de las ciencias de la computación

Aunque los antecedentes del computador se remontan al ábaco (que en su presente forma, fue introducido en China sobre el 1200 d. C.), la tecnología que hoy en día disfrutamos no nació de la noche a la mañana, sino que paso por una serie de etapas, las cuales según algunos autores está formada por cinco generaciones, pero existen algunos que afirman que actualmente nos encontramos en una sexta veamos un resumen de lo que se considera primeramente la prehistoria de las ciencias de la computación.

TABLA 1

PREHISTORIA Y EVOLUCION DE LA COMPUTADORA

EPOCA	MAQUINA	TECNOLOGIA	AUTOR	OPERACIONES
1200 D.C,	Abaco	ABACO	CHINOS	+ - *
1623-1662	Maquina analítica de pascal	RUEDAS DENTADAS	BLAISE PASCAL	+ -
1646-1716	Mejora de la Maquina de pascal	RUEDAS DENTADAS (mejora de pascalina)	LEIBNIZ	+ - * /
1823	Máquina de diferencias analíticas	RUEDAS DENTADAS CON MEMORIA	BABBAGE (padre de la computación)	+ - * raíz cuadrada, polinomios de segundo grado
1885	Máquina del Censo	TARJETAS PERFORADAS	HERMAN HOLLEIRITH	+ - * /
1937	MARK I	ELECTROMECHANICA BASADO EN LA MAQUINA DE BABBAGE	AIKEN	programable
1940	ENIAC	DIGITAL	ECKERT-MAUCHLY	Programable

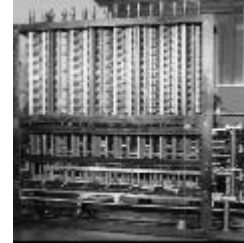
1952	Maquina con memoria	DISPOSITIVOS ELECTROMAGNETI COS	JOHN VON NEWMAN	programable



ABACO



PASCALINA



MAQUINA DE BABBAGE

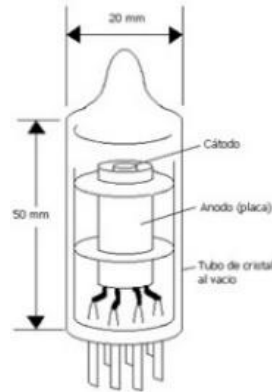
1.1.1. La primera generación de computadoras (1938 – 1952)

La primera generación ocupó la década de los cincuenta. Estas máquinas tenían las siguientes características:

- Usaban tubos al vacío para procesar información.
- Usaban tarjetas perforadas para entrar los datos y los programas.
- Usaban cilindros magnéticos para almacenar información e instrucciones internas.
- Eran sumamente grandes, utilizaban gran cantidad de electricidad, generaban gran cantidad de calor y eran sumamente lentas.
- Se comenzó a utilizar el sistema binario para representar los datos.
- Se evoluciona también los lenguajes de programación.

En esta generación las máquinas son grandes y costosas.

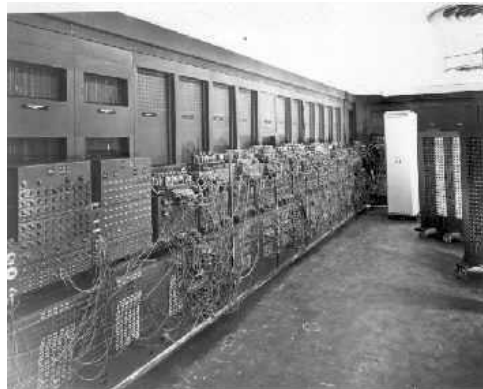
Dibujo simplificado de un tubo de vacío.



Fuente:(Angulo et all, 1995).

En 1946 aparece el primer computador fabricado con Electrónica Digital, el computador ENIAC. La entrada y salida de datos y resultados se realizaban mediante tarjetas perforadas. Tenía unos 18.000 tubos de vacío, pesaba 30 toneladas, ocupaba 1.500 pies cuadrados y realizaba 5.000 sumas por segundo.

Figura 2. ENIAC

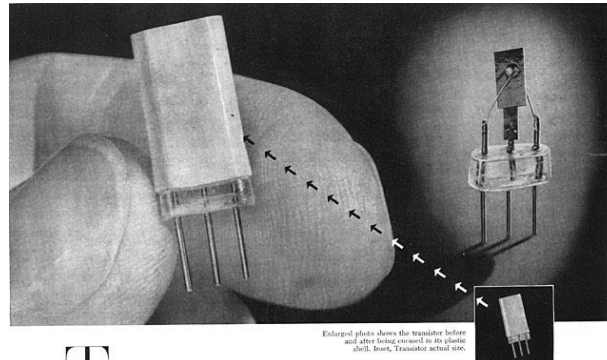


1.1.2. La segunda generación de computadoras (1953 – 1962)

La tecnología de esta generación está caracterizada por el descubrimiento *del transistor*, que se utilizan en la construcción de las Unidades Centrales de Proceso. El transistor, al ser más pequeño, más barato y de menor consumo que la válvula, hizo a los computadores más

asequibles en tamaño y precio. Las memorias empezaron a construirse con núcleos de ferrita. (Angulo 1995).

RCA Junction Transistor Ad (1953)



Transistor
mighty mite of electronics

Las características de la segunda generación son las siguientes:

- Usaban transistores para procesar información.
- Los transistores eran más rápidos, pequeños y más confiables que los tubos al vacío.
- 200 transistores podían acomodarse en la misma cantidad de espacio que un tubo al vacío.
- Usaban pequeños anillos magnéticos para almacenar información e instrucciones. cantidad de calor y eran sumamente lentas.
- Se desarrollaron nuevos lenguajes de programación como COBOL y FORTRAN, los cuales eran comercialmente accesibles.
- Surgieron las minicomputadoras y los terminales a distancia.
- Se comenzó a disminuir el tamaño de las computadoras.



1.1.3. La tercera generación (1964 - 1971)

En esta generación el elemento más significativo es el circuito integrado aparecido en 1964 y consistente en el encapsulamiento de gran cantidad de componentes discretos (resistencias, condensadores, diodos y transistores) conformados uno o varios circuitos en una pastilla.

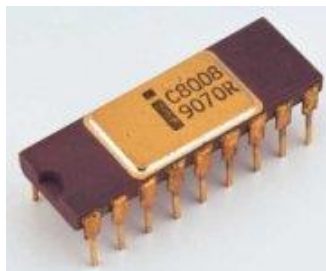
Las principales características de esta generación:

- Se desarrollaron circuitos integrados para procesar información.
- Se desarrollaron los "chips" para almacenar y procesar la información. Un "chip" es una pieza de silicio que contiene los componentes electrónicos en miniatura llamados semiconductores.
- Surge la multiprogramación.
- Las computadoras pueden llevar a cabo ambas tareas de procesamiento o análisis matemáticos.
- Emerge la industria del "software".
- Se desarrollan las minicomputadoras IBM 360 y DEC PDP-1.
- Otra vez las computadoras se tornan más pequeñas, más ligeras y más eficientes. Consumían menos electricidad, por lo tanto, generaban menos calor.

Computadora IBM 360



1.1.4. La cuarta generación (1971-1988)



Aparecen los microprocesadores que es un gran adelanto de la microelectrónica, son circuitos integrados de alta densidad y con una velocidad impresionante. Aquí nacen las computadoras personales que han adquirido proporciones enormes y que han influido en la sociedad en general sobre la llamada "revolución informática".

Características de esta generación:

- Se desarrolló el microprocesador.
- Se colocan más circuitos dentro de un "chip".
- "LSI - Large Scale Integration circuit".
- "VLSI - Very Large Scale Integration circuit".
- Cada "chip" puede hacer diferentes tareas.
- Se reemplaza la memoria de anillos magnéticos por la memoria de "chips" de silicio.
- Se desarrollan las microcomputadoras, o sea, computadoras personales o PC.
- La micro miniaturización de los circuitos electrónicos.

1.1.5. La quinta Generación (1983-20...)

La **quinta generación de computadoras** es como se designa a la etapa que abarca desde el año 1983 hasta la actualidad, un periodo donde las computadoras son veloces y eficientes y poseen un desarrollado software moderno.

Los aparatos electrónicos son pequeños, delgados y fáciles de transportar permitiendo además el almacenamiento de información multimedia en dispositivos sólidos y tarjetas de memoria de gran capacidad.

Dicho proyecto tenía por objetivo elaborar un nuevo tipo de computadoras que fueran capaces de utilizar técnicas y tecnologías de inteligencia artificial, así como de implementar un hardware y software avanzados.

El objetivo era desarrollar máquinas que fueran capaces de procesar el lenguaje natural y alcanzaran habilidades lógicas propias de los seres humanos al resolver problemas de codificación complejos. Todos los idiomas de alto nivel como C y C++, Java, .net, etc., son utilizados en esta generación.

Características principales de la quinta generación de computadoras

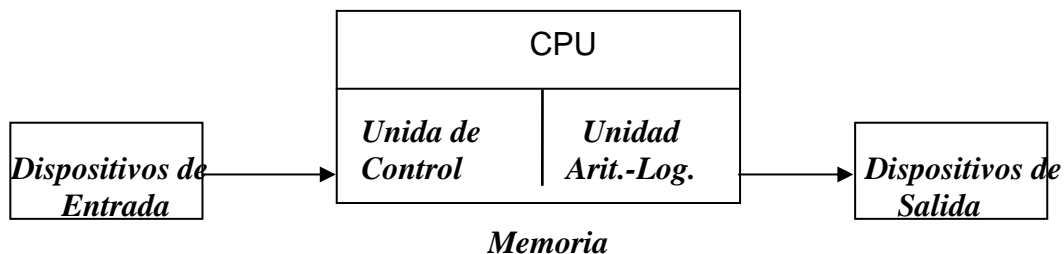
- 1) Estarán hechas con microcircuitos de muy alta integración, que funcionarán con un alto grado de paralelismo y emulando algunas características de las redes neurales con las que funciona el cerebro humano.
- 2) Computadoras con Inteligencia Artificial
- 3) Interconexión entre todo tipo de computadoras, dispositivos y redes (redes integradas)
- 4) Integración de datos, imágenes y voz (entorno multimedia)
- 5) Utilización del lenguaje natural (lenguaje de quinta generación)

1.2 Componentes físicos y lógicos de un computador

1.2.1 Componentes físicos

Denominados Hardware o soporte físico, es el conjunto de elementos materiales que componen un ordenador, incluyen dispositivos electrónicos y electromecánicos, circuitos. Cables, tarjetas y periféricos de todo tipo y otros elementos físicos.

Organización física de una computadora



La computadora está compuesta físicamente por los siguientes elementos:



Organizando todos estos elementos podemos agruparlos en:

1. **Periféricos o Dispositivos de Entrada:** Como su nombre lo indica, sirven para introducir datos en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Ejemplos: teclado, scanner (digitalizadores de rastreo), mouse (ratón), TrackBall (bola de ratón estacionario), joystick (palancas de juego), lápiz óptico, cámaras, etc.
2. **Periféricos o Dispositivos de Salida:** Es lo que la computadora utiliza para comunicarse con nosotros, visualizando información en ella. Regresan los datos procesados que sirven de información al usuario. Ejemplo: monitor, impresora, parlantes.
3. **La Unidad Central de Procesamiento (C.P.U) :** Es el cerebro de la computadora en el que se encuentra la electrónica inteligente del sistema se divide en tres partes:
 - **Unidad de Control:** Coordina las actividades de la computadora y determina que operaciones se deben realizar y en qué orden; así mismo controla la entrada y salida de datos antes que salgan los datos los envía a la memoria y a la UAL para su tratamiento o proceso. Se orienta por las instrucciones del programa.
 - **Unidad Aritmética – Lógica:** Realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.
 - **La Memoria** de la computadora se divide en dos: Memoria Central o Interna y Memoria Auxiliar o Externa.

1.2.2 Componentes lógicos del computador

Se conoce también como **software** al *equipamiento lógico o soporte lógico* de una computadora; comprende el conjunto de los componentes **lógicos** necesarios que hacen posible la realización de tareas específicas.

Definición de software. -Probablemente la definición más formal de software sea la siguiente: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Se puede clasificar al software en tres grandes tipos:

- a. Software de sistema:** Su objetivo es desvincular adecuadamente al usuario y al programador de los detalles de la computadora que se use, aislándolo especialmente del procesamiento referido a las características internas de: memoria, discos, puertos y dispositivos de comunicaciones, impresoras, pantallas, teclados, etc. Incluye entre otros:
- **Sistemas operativos:** Un **sistema operativo** (SO) es el programa o conjunto de programas que efectúan la gestión de los procesos básicos de un sistema informático, y permite la normal ejecución del resto de las operaciones.
 - **Controladores de dispositivos:** Llamado normalmente **controlador** (en inglés, *device driver*) es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz - posiblemente estandarizada- para usarlo.
 - **Herramientas de diagnóstico:** Es un software que permite monitorear y en algunos casos controlar la funcionalidad del hardware, como: computadoras, servidores y periféricos, según el tipo y sus funciones. Estos dispositivos pueden ser, la memoria RAM, el procesador, los discos duros, ruteadores, tarjetas de red, entre muchos dispositivos más.

b. Software de programación: Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluye entre otros:

- **Compiladores:** Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar
- **Intérpretes: intérprete o interpretador** es un programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.
- **Entornos de Desarrollo Integrados (IDE):** Agrupan las anteriores herramientas, usualmente en un entorno visual, de forma tal que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etc. Habitualmente cuentan con una avanzada interfaz gráfica de usuario (GUI).

c. Software de aplicación: Es aquel que permite a los usuarios llevar a cabo una o varias tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios. Incluye entre otros: Aplicaciones para Control de sistemas y automatización industrial.

- **Aplicaciones ofimáticas:** Es una recopilación de aplicaciones, las cuales son utilizados en oficinas y sirve para diferentes funciones como crear, modificar, organizar, escanear, imprimir, etc. archivos y documentos.
- **Software educativo:** Es un software destinado a la enseñanza y el aprendizaje autónomo y que, además, permite el desarrollo de ciertas habilidades cognitivas
- **Software empresarial:** Se entiende generalmente cualquier tipo de software que está orientado a ayudar a una empresa a mejorar su productividad o a medirla.
- **Videojuegos:** Es un software creado para el entretenimiento en general y basado en la interacción entre una o varias personas y un aparato electrónico que ejecuta dicho videojuego; este dispositivo electrónico puede ser una computadora, una videoconsola, un teléfono móvil, los cuales son conocidos como plataformas.

- **Software de Cálculo Numérico y simbólico:** El **análisis numérico** o **cálculo numérico** es la rama de las matemáticas que se encarga de diseñar algoritmos para, a través de números y reglas matemáticas simples, simular procesos matemáticos más complejos aplicados a procesos del mundo real.
- **Software de Diseño Asistido (CAD):** El **diseño asistido por computadora**, más conocido por sus siglas inglesas **CAD** (*computer-aided design*), es el uso de un amplio rango de herramientas computacionales que asisten a ingenieros, arquitectos y a otros profesionales del diseño en sus respectivas actividades.
- **Software de Control Numérico (CAM):** La base de datos que se desarrolla durante el CAD es procesada por el CAM, para obtener los datos y las instrucciones necesarias para operar y controlar la maquinaria de producción, el equipo de manejo de material y las pruebas e inspecciones automatizadas para establecer la calidad del producto.



1.3 Modelado de datos en binario de caracteres y números

Codificación

En un ambiente de sistemas digitales se denomina codificación a la asignación de un significado a una configuración de bits.

Al modelar problemas es usual encontrar variables que pueden tomar múltiples valores, se denomina codificación al proceso de convertir esas variables en valores.

1.3.1. Código BCD

El código ponderado más empleado es el **BCD**, que representa a los dígitos decimales por secuencias de bits en sistema binario. **BCD** es un acrónimo de **Binary Coded Decimal** es de decimal codificado en binario.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

La tabla es el código. Cada secuencia de 4 bits es una palabra del código. A cada dígito decimal se le asocia una secuencia de 4 bits.

La posición de más a la izquierda, o más significativa, tiene peso 8. El bit menos significativo el de más a la derecha, tiene ponderación 1. Las ponderaciones son potencias de 2.

Así, la palabra 7, puede interpretarse según:

$$0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 7$$

Para representar el número decimal 12 se requieren 8 bits, o dos palabras de código.

$$12 = 0001\ 0010$$

1.3.2. Código ASCII

La mayor parte de los dispositivos periféricos empleados en sistemas de computación para comunicar a las personas con las máquinas, permiten representar las letras minúsculas y mayúsculas, dígitos decimales, signos de puntuación y caracteres especiales.

Uno de los códigos más empleados es el código **ASCII**. Éste es un código de 7 bits, más uno de paridad. Permite representar 128 símbolos. Su nombre es una abreviación de "**AmericanStandard Code for Information Interchange**".

Se tienen 32 símbolos de control, símbolos de puntuación, letras y números. Los símbolos asociados a los dígitos decimales están entre 0x30 y 0x39; entonces basta considerar los 4 menos significativos para obtener el código BCD equivalente.

Los códigos binarios de las letras mayúsculas y minúsculas difieren en el estado de un bit.

TABLA ASCII

H		D	H		D	H		D	H		D	H		D	H		D	H		D	H		D
00	NULL	00	10	DEL	16	20		32	30	0	48	40	@	64	50	P	80	60	`	96	70	p	112
01	SOH	01	11	DC1	17	21	!	33	31	1	49	41	A	65	51	Q	81	61	a	97	71	q	113
02	STX	02	12	DC2	18	22	"	34	32	2	50	42	B	66	52	R	82	62	b	98	72	r	114
03	EXT	03	13	DC3	19	23	#	35	33	3	51	43	C	67	53	S	83	63	c	99	73	s	115
04	EOT	04	14	DC4	20	24	\$	36	34	4	52	44	D	68	54	T	84	64	d	100	74	t	116
05	ENQ	05	15	NAK	21	25	%	37	35	5	53	45	E	69	55	U	85	65	e	101	75	u	117
06	ACK	06	16	SYN	22	26	&	38	36	6	54	46	F	70	56	V	86	66	f	102	76	v	118
07	BEL	07	17	ETB	23	27	'	39	37	7	55	47	G	71	57	W	87	67	g	103	77	w	119
08	BS	08	18	CAN	24	28	(40	38	8	56	48	H	72	58	X	88	68	h	104	78	x	120
09	TAB	09	19	EM	25	29)	41	39	9	57	49	I	73	59	Y	89	69	i	105	79	y	121
0a	LF	10	1a	SUB	26	2a	*	42	3a	:	58	4a	J	74	5a	Z	90	6a	j	106	7a	z	122
0b	VT	11	1b	ESC	27	2b	+	43	3b	;	59	4b	K	75	5b	[91	6b	k	107	7b	{	123
0c	FF	12	1c	FS	28	2c	,	44	3c	<	60	4c	L	76	5c	\	92	6c	l	108	7c		124
0d	CR	13	1d	GS	29	2d	-	45	3d	=	61	4d	M	77	5d]	93	6d	m	109	7d	}	125
0e	SO	14	1e	RS	30	2e	.	46	3e	>	62	4e	N	78	5e	^	94	6e	n	110	7e	~	126
0f	SI	15	1f	US	31	2f	/	47	3f	?	63	4f	O	79	5f		95	6f	o	111	7f	del	127

1.4 Conversión de sistemas de numeración

1.4.1. Sistema decimal

Su origen lo encontramos en la India y fue introducido en España por los árabes. Su base es 10. Emplea 10 caracteres o dígitos diferentes para indicar una determinada cantidad: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. El valor de cada símbolo depende de su posición dentro de la cantidad a la que pertenece. Veámoslo con un ejemplo:

$$136_{10} = 1 \cdot 10^2 + 3 \cdot 10^1 + 6 \cdot 10^0$$

1.4.2. Sistema binario

Es el sistema digital por excelencia, aunque no el único, debido a su sencillez. Su base es 2. Emplea 2 caracteres: 0 y 1. Estos valores reciben el nombre de bits (dígitos binarios). Así, podemos decir que la cantidad 10011 está formada por 5 bits. Veamos con un ejemplo como se representa este número teniendo en cuenta que el resultado de la expresión polinómica dará su equivalente en el sistema decimal:

$$10011_2 = 1 \cdot 10^4 + 0 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 1 \cdot 10^0 = 19_{10}$$

1.4.3. Sistema octal

Posee ocho símbolos: 0, 1, 2, 3, 4, 5, 6, 7. Su base es 8. Este sistema tiene una peculiaridad que lo hace muy interesante y es que la conversión al sistema binario resulta muy sencilla ya que, $8 = 2^3$.

1.4.4. Sistema hexadecimal

Está compuesto por 16 símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Su base es 16. Es uno de los sistemas más utilizados en electrónica, ya que además de simplificar la escritura de los números binarios, todos los números del sistema se pueden expresar en cuatro bits binarios al ser $16 = 2^4$.

CONVERSIONES

Conversión de un sistema de cualquier base r a base 10

Utilizando la notación polinomial:

Ejemplo:

$$(10100)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = (20)_{10}$$

Conversión de un sistema de base 10 a base r

Utilizando el método de las divisiones sucesivas por la base:

Ejemplo:

$$\begin{array}{r}
 65029 \quad | \quad 16 \\
 \hline
 5 \quad 4064 \quad | \quad 16 \\
 \hline
 \downarrow \quad 0 \quad 254 \quad | \quad 16 \\
 \downarrow \quad \downarrow \quad 14 \quad 15 \\
 \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
 5 \quad 0 \quad E \quad F \longrightarrow FE05
 \end{array}$$

1.4.5. Unidades de Almacenamiento

Nombre	Símbolo	Potencias binarias y valores decimales
byte	b	$2^0 = 1$
Kbyte	KB	$2^{10} = 1\,024$
Megabyte	MB	$2^{20} = 1\,048\,576$
Gigabyte	GB	$2^{30} = 1\,073\,741\,824$
Terabyte	TB	$2^{40} = 1\,099\,511\,627\,776$
Petabyte	PB	$2^{50} = 1\,125\,899\,906\,842\,624$
Exabyte	EB	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
Zettabyte	ZB	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$
Yottabyte	YB	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$

1.5 Definición de algoritmos

La palabra algoritmo proviene del nombre del matemático de origen árabe "Al-Khwarizmi", sobrenombre del célebre matemático Mohamed ben Musa.

Un algoritmo es una secuencia ordenada de pasos, bien definidos, que permite la resolución de un problema o realizar una tarea específica.

1.6 Características de los Algoritmos.

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo **debe ser preciso**, importa el orden de realización de cada paso.
- Un algoritmo **debe estar definido**. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo **debe ser finito**. el algoritmo se debe terminar en algún momento; o sea, debe tener un número finito de pasos.
- Un algoritmo **debe ser simple**: fácil de entender.

1.7 Lenguajes de programación para la representación de algoritmos

Es un conjunto de símbolos, caracteres y reglas (programas) que permiten a las personas comunicarse con la computadora. Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, calculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación se clasifican en:

- **Lenguaje Maquina:** Son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria más pequeña el bit (dígito binario 0 o 1).
- **Lenguaje de Bajo Nivel (Ensamblador):** En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.
- **Lenguaje de Alto Nivel:** Los lenguajes de programación de alto nivel (VISUAL BASIC, PASCAL, C++,JAVA, etc.) son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos (en general en inglés), lo que facilita la escritura y comprensión del programa.



1.8 Ejercicios Propuestos

- A) Realice un listado de los componentes hardware y software de un computador.
- B) Realice un esquema de las generaciones de las computadoras
- C) Convierta los números decimales al sistema binario 56, 77, 98 y viceversa.
- D) Convierta los números decimales a octal 1205, 59, 125 y viceversa.
- E) Convierta los números decimales a hexadecimal 781, 482, 78 y viceversa.

II. ESTRUCTURAS SECUENCIALES.

2.1 Metodología de resolución de problemas

Las Fases de resolución de un problema con computadora son:

Definición del Problema. - Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa, se debe tener un conocimiento base.

a) Análisis del Problema. - Una vez que se ha comprendido lo que se desea que haga la computadora, es necesario definir:

- Los datos de entrada.
- Cuál es la información que se desea producir (salida)
- Los métodos y fórmulas que se necesitan para procesar los datos.

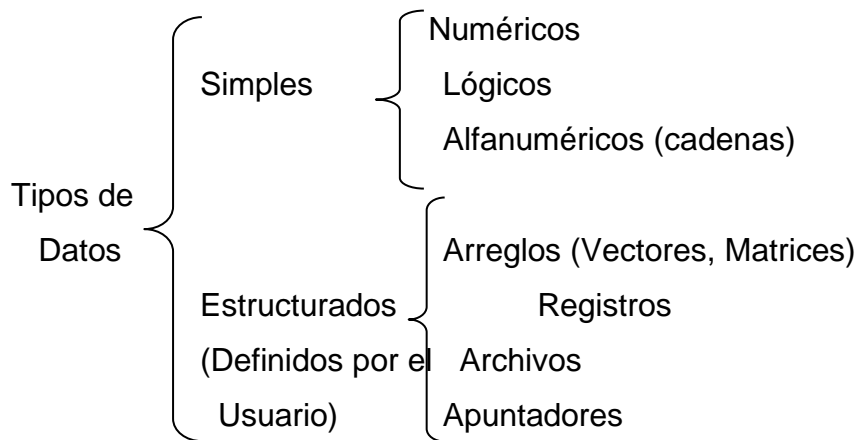
b) Diseño del Algoritmo. - Se determina el cómo se desarrollara el programa, es decir se dará respuesta al cómo? se resolverá el problema propuesto.

c) Codificación. - La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del algoritmo que puede estar representado en diagrama de flujo o pseudocódigo), en una sucesión de instrucciones detalladas, en un código reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

d) Prueba y Depuración. - Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama **depuración**.

2.2 Tipos de datos numéricos, lógicos y alfanuméricos.

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'a', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.



Tipos de Datos Simples

- **Datos Numéricos:** Permiten representar valores escalares de forma numérica, esto incluye a los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.

Ejemplo: 56, -1, 290,3.1416 también 5gr, 25 mts.

- **Datos Lógicos:** Son aquellos que solo pueden tener dos valores (verdadero o falso) ya que representan el resultado de una comparación entre otros datos (numéricos o alfanuméricos).

Ejemplo:

$F \text{ AND } V = F$ también $V \text{ OR } F = V$

- **Datos Alfanuméricos (caracteres numéricos y alfabéticos) (Cadenas):** Es una secuencia de caracteres alfanuméricos que permiten representar valores de forma descriptiva, esto incluye nombres de personas, direcciones, etc. Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no es posible hacer operaciones con ellos. Este tipo de datos se representan encerrados entre comillas.

Ejemplo:

“Av. Del Ejercito”

“1997”

“4434325”

“Pedro Milanés”

2.3 Variables y Constantes

a) Variable: Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante el desarrollo del algoritmo. Para poder reconocer una variable en la memoria del computador, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

Las variables tienen las siguientes características:

- Un nombre o identificador con el que nos referimos en el algoritmo a dicha variable.
- Un tipo, que indica los posibles valores que puede tomar la variable y las acciones que son posibles con ella.
- Un valor que está contenido en la memoria del ordenador.
- Una dirección de memoria que indica dónde está localizada la variable en la memoria.

Ejemplos de Variables

IVA=0.15 ; promedio=7,6 ; costo=2500 → Numéricas

Letra= "a"; Apellido= "López"; Dirección= "Av. Libertador #190" → Alfanuméricas

Bandera=Falso → Lógico

Suma = a + b/c

b) Constante: Una constante es un dato numérico o alfanumérico que no cambia durante el desarrollo del algoritmo.

Ejemplo:

PI = 3,1416

Mayor= -999

Color= "Rojo"

Ejemplo:

Área = pi * radio ^ 2

Las variables son: el radio, el área y la constante es pi

2.4 Expresiones aritméticas, lógicas y relacionales

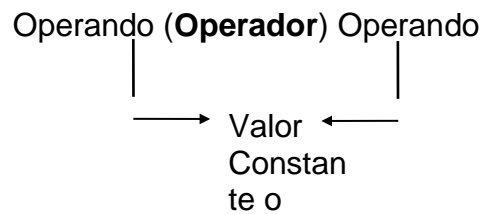
Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

Por ejemplo:

$a + (b + 3) / c ; \quad 3 <> 20 ; \quad (a = b) \text{ and } (3 > 5)$

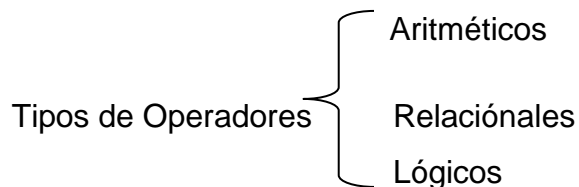
Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Una expresión consta de operadores y operandos como se ve en el siguiente esquema:



Operandos: Puede ser una constante o una variable son los distintos valores que puede tomar una variable.

Operadores: Son elementos que relacionan de forma diferente, los valores de una o más variables y/o constantes. Es decir, los operadores nos permiten manipular valores.



1) Operadores Aritméticos: Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes). Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales.

Operadores Aritméticos

+	Suma
-	Resta
*	Multiplicación
/	División

Mod Modulo (residuo de la división entera)

Div División entera

^ Exponenciación

Ejemplos:

Expresión	Resultado
-----------	-----------

5 / 2	= 2.5
-------	-------

15 Mod 2	= 1
----------	-----

6 + 20 / 5 - 8	= 2
----------------	-----

Prioridad de los Operadores Aritméticos

- Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero.

- Dentro de una misma expresión los operadores se evalúan en el siguiente orden.

1.- ^ Exponenciación

2.- *, /, div, mod Multiplicación, División y Modulo

3.- +, - Suma y resta.

Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

2)Operadores Relacionales:

- Se utilizan para establecer una relación entre dos valores.
- Compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).
- Los operadores relacionales comparan valores del mismo tipo (numéricos o cadenas)
- Tienen el mismo nivel de prioridad en su evaluación.

Operadores Relacionales

>	Mayor que
<	Menor que
> =	Mayor o igual que
< =	Menor o igual que
< >	Diferente
=	Igual

Ejemplos:

Si $a = 5$ $b = 10$ $c = 25$

$a + b > c$ Falso

$a - b < c$ Verdadero

$b - a + 20 = c$ Falso

$a * b < > c$ Verdadero

Operadores Lógicos:

- Estos operadores se utilizan para establecer relaciones entre valores lógicos.
- Estos valores pueden ser resultado de una expresión relacional. Estos son:

And	Y
Or	O
Not	Negación

Operador And

<i>Operando1</i>	<i>Operador</i>	<i>Operando2</i>	<i>Resultado</i>
T	AND	T	T
T	AND	F	F
F	AND	T	F
F	AND	F	F

Operador Or

<i>Operando1</i>	<i>Operador</i>	<i>Operando2</i>	<i>Resultado</i>
T	OR	T	T
T	OR	F	T
F	OR	T	T
F	OR	F	F

Operador Not

<i>Operando</i>	<i>Resultado</i>
T	F
F	T

Ejemplos:

Para $a=10$; $b=20$; $c=30$

$(a < b)$ and $(b < c)$

$(10 < 20)$ and $(20 < 30)$

T and T
| |
→ T ←

Prioridad de los Operadores en General

1. - ()

2. - ^

3. - *, /, Mod, Div, Not

4. - +, -, And

5. - >, <, > =, < =, < >, =, Or

Ejemplos:

Sean $a = 10$, $b = 12$, $c = 13$ y $d = 10$

1) $((a > b) \text{ or } (a < c)) \text{ and } ((a = c) \text{ or } (a > = b))$

(F or V) and (F or F)

$\boxed{\text{V}}$ and $\boxed{\text{F}}$

$\boxed{\text{F}}$

2) $((a > = b) \text{ or } (a < d)) \text{ and } ((a > = d) \text{ and } (c > d))$

(F or F) and (V and V)

$\boxed{\text{F}}$ and $\boxed{\text{V}}$

$\boxed{\text{F}}$

3) not $(a = c) \text{ and } (c > b)$

$\begin{array}{cc} \text{F} & \text{V} \\ | & | \\ & \text{T} \end{array}$

2.5 Asignaciones

La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor.

El formato general de una operación de asignación es:

Nombre de la variable = variable, constante, expresión

En las asignaciones no se pueden asignar valores a una variable de un tipo diferente del suyo.

Ejemplos:

Suma = a+b

Valor = 2345

Nombre = N

2.6 Entrada/Salida de Datos

-Entrada (Lectura): La lectura consiste en recibir desde un dispositivo de entrada (por ejemplo el teclado) un valor. Esta operación se representa en un pseudocódigo como sigue:

Leer a

Leer b

Donde “a” y “b” son las variables que recibirán los valores

-Salida (Escritura): Consiste en mandar por un dispositivo de salida (por ejemplo monitor o impresora) un resultado (valor de una variable) o mensaje. Este proceso se representa en un pseudocódigo como sigue:

Escribir <mensaje> o Escribir <variable>

Escribir “El resultado es:”, R

Donde “El resultado es:” es un mensaje que se desea aparezca y R es una variable que contiene un valor.

2.7 Representación en Pseudocódigo y diagrama de flujo

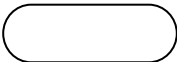
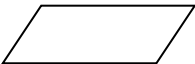
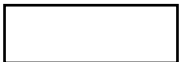
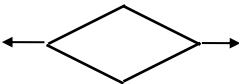

Las tres herramientas utilizadas comúnmente para diseñar algoritmos son:

Diagrama de Flujo y Pseudocódigo

- Diagrama de Flujo

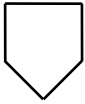
Un diagrama de flujo es la representación gráfica de un algoritmo. Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar los procesos.

Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI).

<u>SÍMBOLO</u>	<u>DESCRIPCIÓN</u>
	Indica el inicio y el final de nuestro diagrama de flujo.
	Indica la entrada y salida de datos.
	Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.
	Símbolo de decisión indica la realización de una comparación de valores.
	Se utiliza para representar los subprogramas.



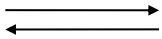
Conector dentro de página. Representa la continuidad del diagrama dentro de la misma página.



Conector fuera de página. Representa la continuidad del diagrama en otra página.



Para la declaración de variable



Líneas de flujo o dirección. Indican la

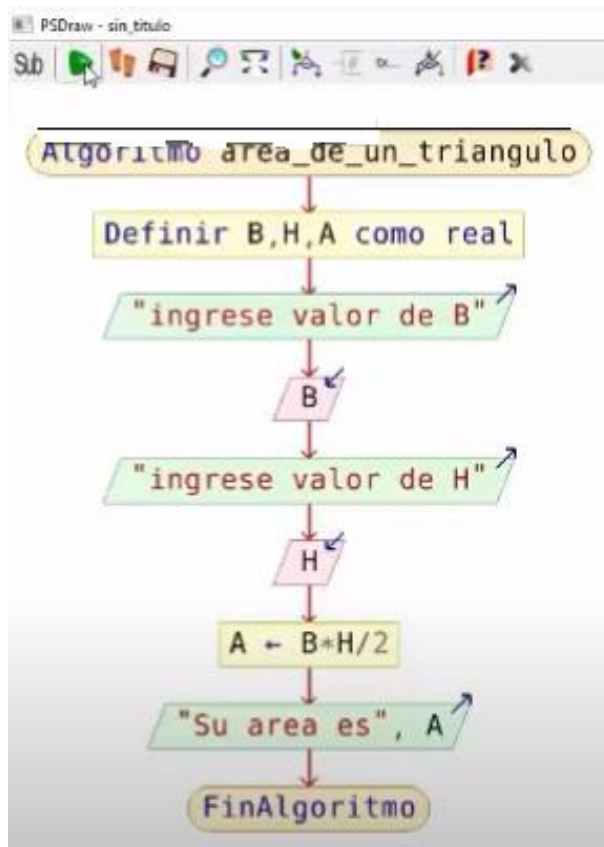


Secuencia en que se realizan las operaciones.

Recomendaciones para el diseño de Diagramas de Flujo

- Se deben usar solamente líneas de flujo horizontal y/o vertical.
- Se debe evitar el cruce de líneas utilizando los conectores.
- Se deben usar conectores solo cuando sea necesario.
- No deben quedar líneas de flujo sin conectar.
- Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

Un Ejemplo:



- **Pseudocódigo**

Es una mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El Pseudocódigo utiliza palabras que indican el proceso a realizar.

A.27 lista de variables
 * + = < operadores y funciones

```

1  Proceso sin_titulo
2
3
4      Escribir "Escribe un valor en la variable 1"
5      leer var1
6
7      Escribir "Escribe un valor en la variable 2"
8      leer var2
9
10     suma=var1+var2
11     resta=var1-var2
12     multiplicacion=var1*var2
13     division=var1/var2
14
15     escribir "La suma es: ", suma
16     escribir "La resta es: ", resta
17     escribir "La multiplicacion es: ", multiplicacion
18     escribir "La division es: ", division
19
20 FinProceso
21

```

Ventajas de utilizar un Pseudocódigo

- Ocupa menos espacio en una hoja de papel.
- Permite representar en forma fácil operaciones repetitivas complejas.
- Es muy fácil pasar de pseudocódigo a un programa en algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

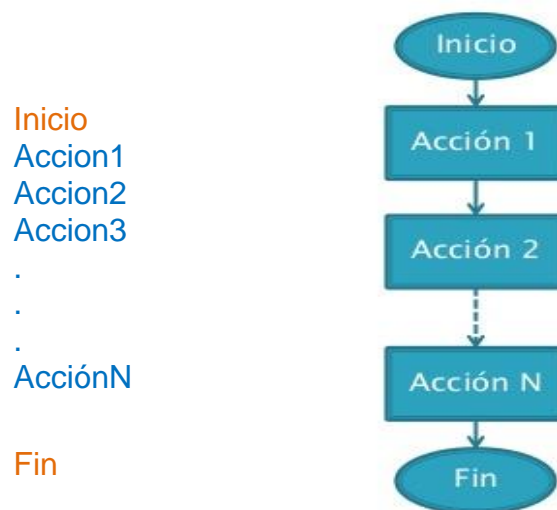
2.8 Concepto de Secuencia

Una secuencia es una serie de acciones que se realizan una tras otra siguiendo un orden para realizar una tarea específica o resolver un problema.

Características principales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.

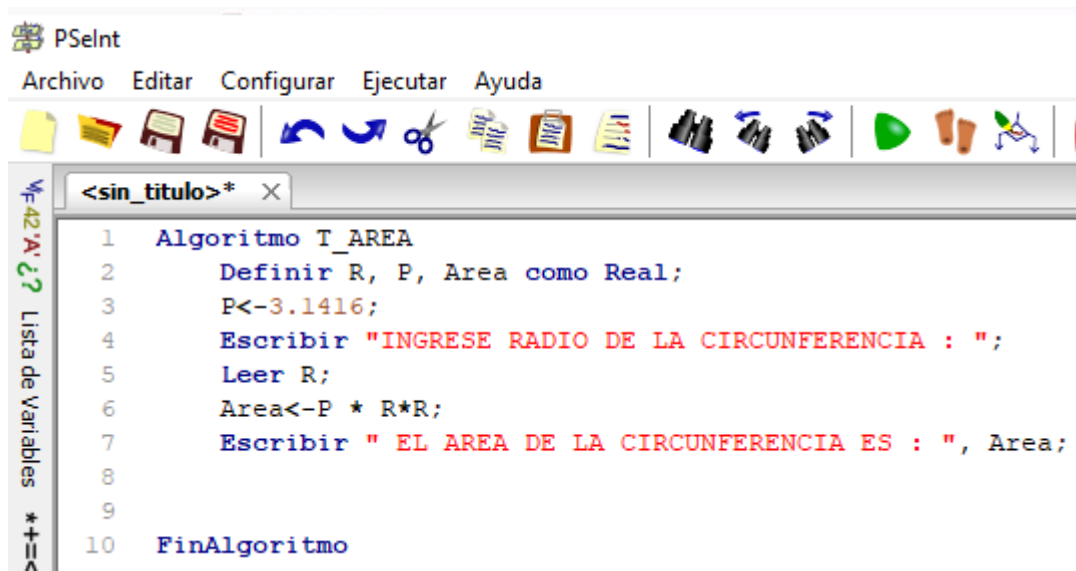
Una estructura secuencial se representa de la siguiente forma:



Ejemplos de algoritmos secuenciales

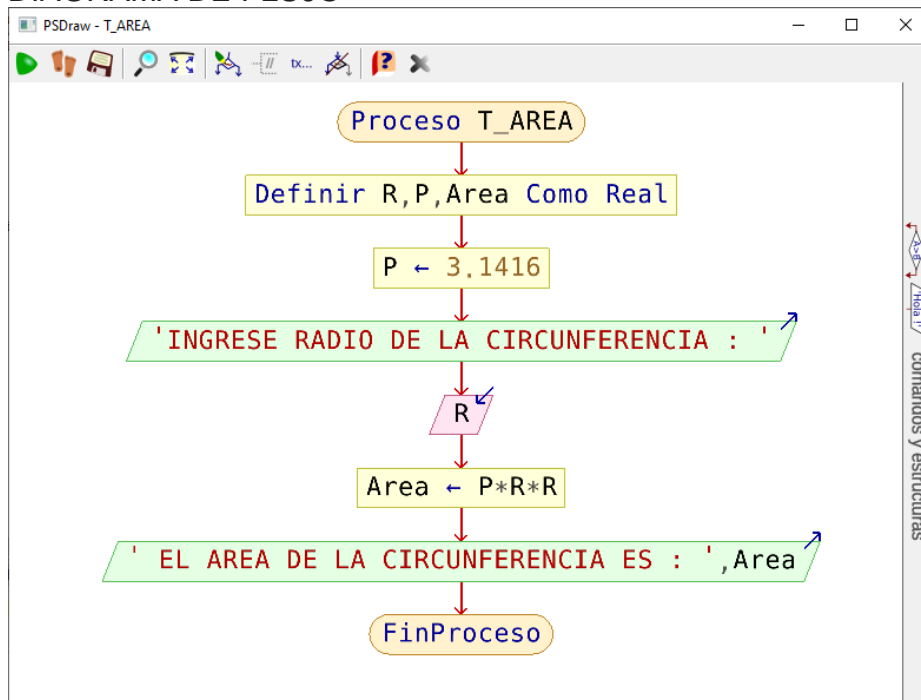
1. Se requiere obtener el área de una circunferencia. Realizar el algoritmo correspondiente y representarlo mediante un diagrama de flujo y el pseudocódigo correspondiente.

PSEUDOCODIGO



```
1  Algoritmo T_AREA
2      Definir R, P, Area como Real;
3      P<-3.1416;
4      Escribir "INGRESE RADIO DE LA CIRCUNFERENCIA : ";
5      Leer R;
6      Area<-P * R*R;
7      Escribir " EL AREA DE LA CIRCUNFERENCIA ES : ", Area;
8
9
10 FinAlgoritmo
```

DIAGRAMA DE FLUJO



2. Un vendedor recibe un sueldo base más un 15% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

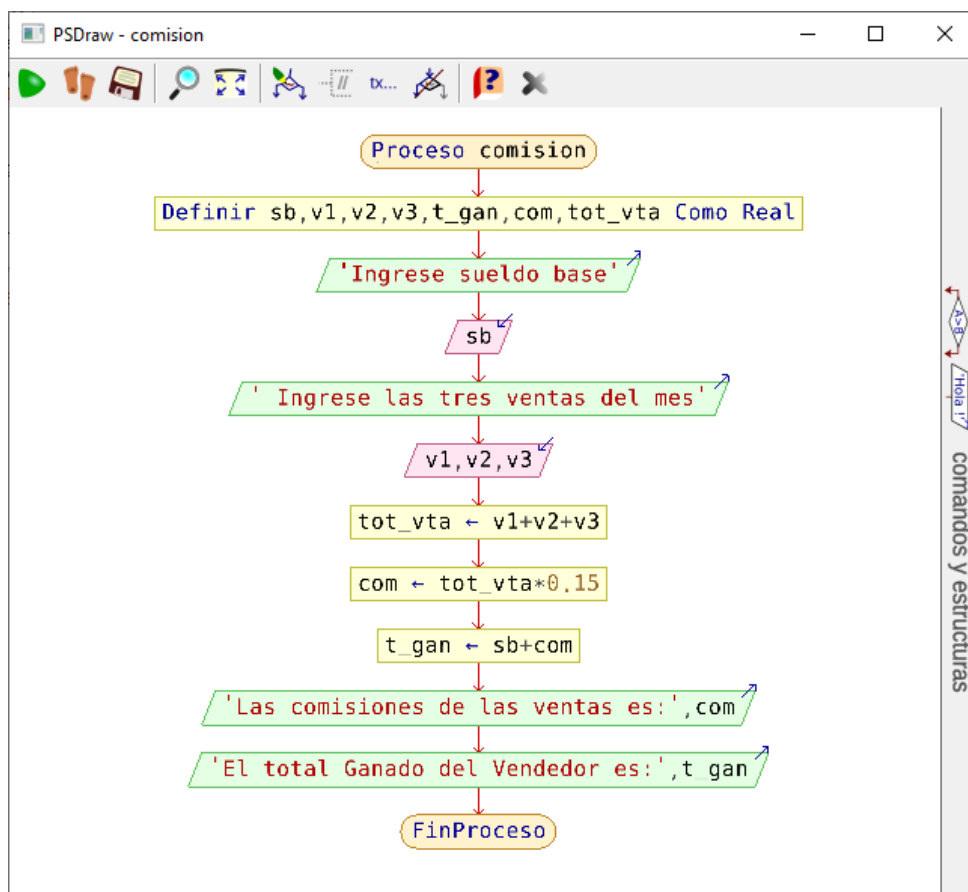
PSEUDOCODIGO

The screenshot shows the PSeInt application window. The menu bar includes Archivo, Editar, Configurar, Ejecutar, and Ayuda. The toolbar contains icons for file operations and execution. The main editor displays the following pseudocode:

```
1  Algoritmo comision
2  Definir sb,v1,v2,v3, t_gan,com,tot_vta como Real;
3  Escribir "Ingrese sueldo base";
4  Leer sb;
5  Escribir " Ingrese las tres ventas del mes";
6  Leer v1, v2, v3;
7  tot_vta <- v1 + v2 + v3;
8  com <- tot_vta * 0.15;
9  t_gan <-sb + com;
10 Escribir "Las comisiones de las ventas es:", com;
11 Escribir "El total Ganado del Vendedor es:", t_gan;
12
13
14 FinAlgoritmo
```

On the left side, there are panels for 'Lista de Variables' and 'Operadores y Funciones'. On the right side, there are panels for 'Comandos y Estructuras' and 'Ejecución'. At the bottom, a status bar indicates 'El pseudocódigo está siendo ejecutado.'

DIAGRAMA DE FLUJO



2.9 Ejercicios Propuestos

1. Realizar un algoritmo para Freír un huevo.
2. Realizar un algoritmo para Hacer pipocas en una olla puesta a fuego.
3. Realizar un algoritmo para invitar a un amigo(a) al cine.
4. Realizar un algoritmo para calcular la hipotenusa de un triángulo rectángulo
5. Realizar un algoritmo para determinar cuánto dinero ahorra una persona en un año si se considera que cada semana ahorra 15% de su sueldo(considerar cuatro semanas por mes y que no cambia el sueldo).
6. Realizar un algoritmo para calcular el promedio que obtendra un estudiante considerando que realiza tres examenes, de los cuales el primero y el segundo tenen una ponderacion de 25% minetras que el tercero de 50%

III. ESTRUCTURAS SELECTIVAS.

Definición

En programación, la estructura de selección es un tipo de estructura de control. También llamada estructura de decisión o estructura selectiva.

En una estructura de selección/decisión, el algoritmo al ser ejecutado toma una decisión, ejecutar o no ciertas instrucciones si se cumplen o no ciertas condiciones. Las condiciones devuelven un valor, verdadero o falso, determinado así la secuencia a seguir.

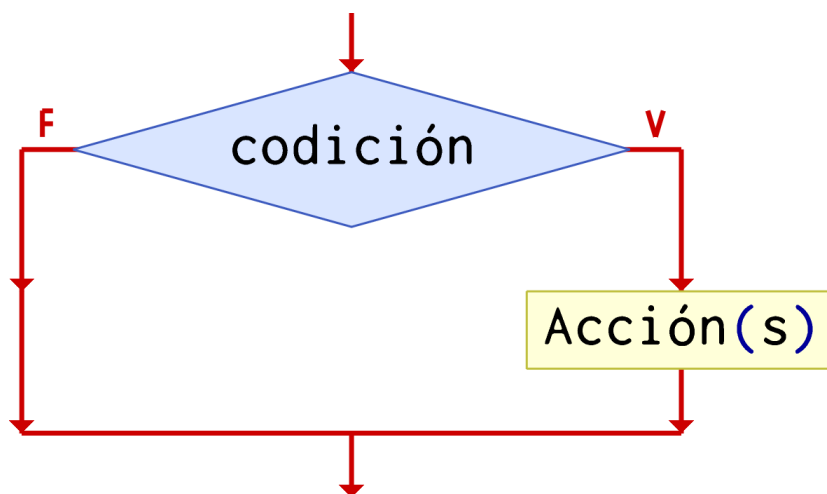
3.1 Estructura de un Algoritmo de Selección

Las estructuras de selección comparan una variable contra otro(s) valor(es), para que, en base al resultado de esta comparación, se siga un curso de acción dentro del algoritmo. La comparación se puede hacer contra otra variable o contra una constante, según se necesite.

Existen tres tipos, las simples, dobles y las múltiples.

3.2 Estructura de selección Simple:

Las estructuras de selección simples se les conocen como “Tomas de decisión”. Estas tomas de decisión tienen la siguiente forma:



Si codición **Entonces**

Acción(s)

FinSi

Ejemplo

Realizar un algoritmo para: Una librería realiza un descuento de 10% por una compra mayor a 500 Bs se desea saber cuánto será el descuento y el total a pagar por la compra.
Algoritmo CondicionSimple

definir compra Como Real

definir total Como Real

escribir "Ingrese monto de compra:"

Leer compra

total = compra

Si (compra > 500) Entonces

total = compra - compra * 0.15

FinSi

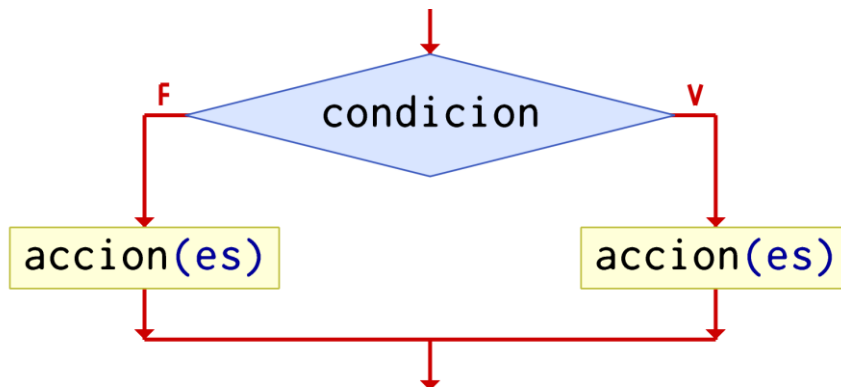
Mostrar "Total a pagar es:", total

FinAlgoritmo

3.3 Estructura de selección Doble:



Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:



Si condición Entonces

Acción(es)

Sino

Acción(es)

FinSi

Donde:

SiIndica el comando de comparación

Condición.....Indica la condición a evaluar
Entonces.....Precede a las acciones a realizar cuando se cumple la condición
Acción(es).....Son las acciones a realizar cuando se cumple o no la condición
Sino.....Precede a las acciones a realizar cuando no se cumple la condición
FinSi.....Indica el final del Si

Ejemplo

Realice un algoritmo para determinar cuánto se debe pagar por una cantidad de lápices considerando que si son 1000 o más el costo es de 0,80 ctvs.; de lo contrario, el precio es de 90 ctvs.

Algoritmo seleccionDoble

```
definir cant_lapiz Como Entero

definir total_pagar como real

Mostrar "Cuantos lápices comprara?"

Leer cant_lapiz

Si (cant_lapiz >= 1000) entonces

    total_pagar <- cant_lapiz*0.80

Sino

    total_pagar <- cant_lapiz*0.90

finSi

Mostrar "El total a pagar por los lápices es:", total_pagar
```

FinAlgoritmo

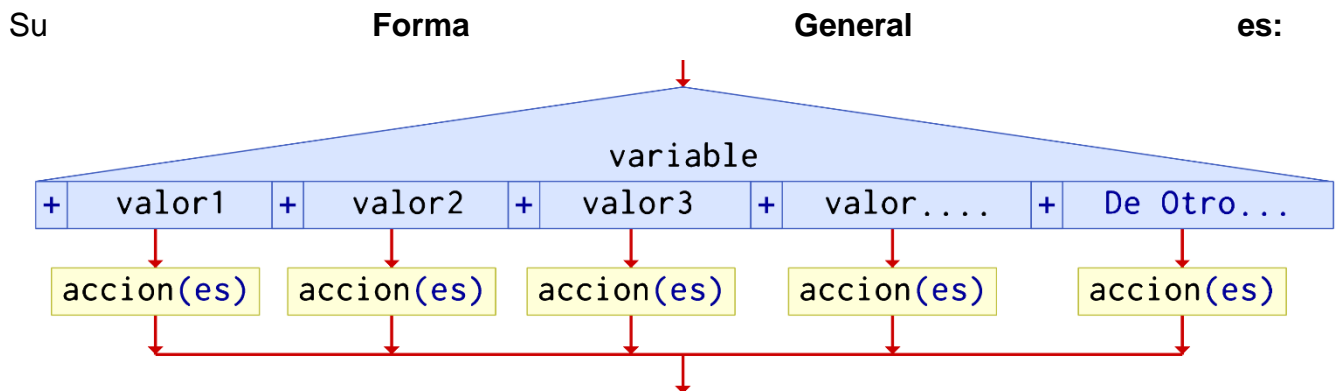
3.4 Estructuras de selección Compuestas

En aquellos problemas en donde un bloque condicional incluye otro bloque condicional se

dice que un bloque está anidado dentro del otro. A este tipo de estructuras se les conoce también como estructuras selectivas compuestas.

3.5 Estructura de selección Múltiple

Las estructuras de comparación múltiples son tomas de decisión especializada que permiten comparar unas variables con distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:



Según *Variable* **hacer**

Op1: Acción(es)

Op2: Acción(es)

OpN: acción(es)

De otro modo

Acción(es)

FinSegun

Ejemplo

Desarrollar un algoritmo que recibe como dato un número entero entre 1 y 12 y escribe el mes correspondiente.

Algoritmo seleccionMultiple

Escribir 'Ingrese un numero entre 1-12'

Leer mes

Segun mes Hacer

1:

Escribir 'enero'

2:

Escribir 'febrero'

3:

Escribir 'marzo'

4:

Escribir 'abril'

5:

Escribir 'mayo'

6:

Escribir 'junio'

7:

Escribir 'julio'

8:

Escribir 'agosto'

9:

Escribir 'septiembre'

10:

Escribir 'octubre'

11:

Escribir 'noviembre'

12:

Escribir 'diciembre'

De Otro Modo:

Escribir 'numero fuera de rango'

FinSegun

FinAlgoritmo

3.6 Estructuras selectivas anidadas

Si <condición> **entonces**

Acción(es)

Sino

Si <condición> **entonces**

Acción(es)

Sino

.

.

Varias
Condiciones

.

FinSi

FinSi

Ejemplo

Realizar un algoritmo para leer 2 números, si los números son iguales multiplicarlos, si el primero es mayor que el segundo restarlos y si no sumarlos que.

Algoritmo Anidadas

definir num1, num2 Como Entero

definir resul Como Entero

Mostrar 'Ingrese dos números: '

Leer num1, num2

Si (num1 = num2) entonces

 resul = num1 * num2

Sino

 Si (num1 > num2) entonces

 resul = num1 - num2

 Sino

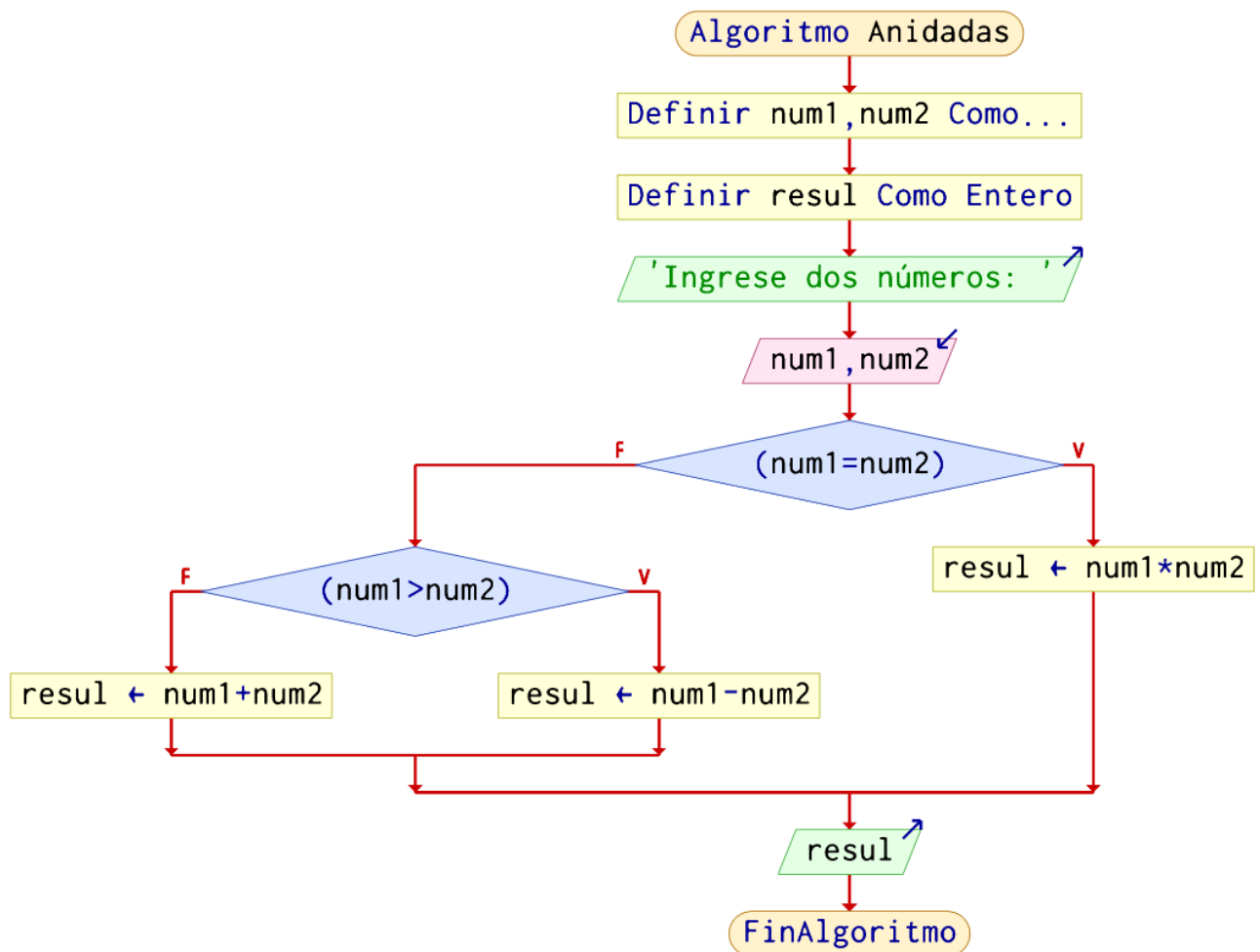
 resul = num1 + num2

FinSi

finSi

Mostrar resul

FinAlgoritmo



3.7 Prueba de escritorio de la solución y depuración

La prueba de escritorio es una herramienta útil para entender que hace un determinado algoritmo, o para verificar que un algoritmo cumple con la especificación sin necesidad de ejecutarlo.

Básicamente, una prueba de escritorio es una ejecución 'a mano' del algoritmo, por lo tanto se debe llevar registro de los valores que va tomando cada una de las variables involucradas en el mismo.

Ejemplo

A continuación, se muestra un ejemplo de prueba de escritorio del siguiente algoritmo para conversión de metros a pies. Donde 1 m = 3.2808 pies.

Algoritmo PruebaEscritorio

definir M, Pies Como Real

Mostrar 'Ingrese los metros'

Leer M

$Pies = M * 3.2808$

Mostrar 'La cantidad de pies es:', Pies

FinAlgoritmo

Prueba de escritorio:

M	Pies	Mensaje
2	$2 * 3.2808 = 6.5616$	La cantidad de pies es: 6.5616
3	$3 * 3.2808 = 9.8424$	La cantidad de pies es: 9.8424

3.8 Aplicaciones al algebra y cálculo

1) Realizar un algoritmo para leer tres números y determinara el mayor de ellos.

Algoritmo elMayor

definir A, B, C Como Entero

definir M Como Entero

Mostrar 'Ingrese tres números: '

Leer A, B, C

Si $(A > B)$ Entonces

Si $A > C$ Entonces

$M \leftarrow A$

Sino

$M \leftarrow C$

FinSi

Sino

Si $B > C$ Entonces

$M \leftarrow B$

Sino

$M \leftarrow C$

FinSi

FinSi

Mostrar 'El mayor de los números es:', M

FinAlgoritmo

2) Una compañía de paquetería internacional tiene servicio en algunos países de América del Norte, América Central, América del Sur, Europa y Asia. El costo por el servicio de paquetería se basa en el peso del paquete y la zona a la que va dirigido. Lo anterior se muestra en la tabla

Costos por el servicio de paquetería

Zona	Ubicación	Costo/gramo
1	América del Norte	\$ 11
2	América Central	\$ 10

3	América del Sur	\$ 12
4	Europa	\$ 24
5	Asia	\$ 27

Parte de su política implica que los paquetes con un peso superior a 5 kg no son transportados, esto por cuestiones de logística y de seguridad. Realicé un algoritmo para determinar el cobro por la entrega de un paquete o, en su caso, el rechazo de la entrega; represéntelo mediante.

Algoritmo cobroPaquete

definir NZ, PE, CO Como Real

Mostrar 'Ingrese Numero de Zona 1 América del Norte; 2 América Central; América del Sur; Europa; Asia'

Leer NZ

Mostrar 'Ingrese peso del paquete en gramos'

Leer PE

Si PE > 5000 Entonces

Mostrar 'No se puede dar el servicio'

Sino

Según NZ hacer

1: CO = PE * 11

2: CO = PE * 10

3: CO = PE * 12

4: CO = PE * 24

De Otro Modo:

$$CO = PE * 27$$

FinSegun

Mostrar “El costo del servicio es”, CO

FinSi

FinAlgoritmo

3.9 Ejercicios Propuestos

- 1) Realizar un algoritmo para determinar si una persona puede votar con base en su edad en las próximas elecciones.
- 2) Realizar un algoritmo para el 21 de septiembre una persona desea comprarle un regalo al ser querido que más aprecia en ese momento, el problema radica en qué regalo puede hacerle en función al dinero con el que cuenta, nlas alternativas que tiene son las siguientes:

REGALO	COSTO
Tarjeta	10 o menos
Chocolates	11 a 100
Flores	101 a 250
Anillo Más de	251

- 3) Realizar un algoritmo para leer números enteros entre 1 y 10 y convertirlos a romanos. Representar en pseudocódigo y diagrama de flujo.
- 4) Diseñar un algoritmo que determine el valor de la factura de la luz de un cliente. El valor de la factura es la suma de la cuota fija (50 Bs.) más una cuota variable que depende del consumo y se calcula así:

Si ha consumido menos de 100 kw, el valor del kilovatio es de 6 Bs.

Si ha consumido entre 100 y 350 kw, el valor del kilovatio es de 9 Bs.

Si ha consumido entre 360 y 600 kw, el valor del kilovatio es de 12 Bs.

Si ha consumido más de 1200 kw, el valor del kilovatio es de 15 Bs. + una multa de 100 Bs
- 5) Leer una vocal en minúscula y convertirla a mayúscula.

IV. ALGORITMOS REPETITIVOS

4.1 Concepto de Bucle e iteración

Son algoritmos en cuya solución se utilizan un mismo conjunto de acciones que se pueden ejecutar una cantidad específica de veces. La cantidad de veces puede ser fija o variable.

El conjunto de acciones que se repiten se denominan bucle (Loop) o ciclo. Un bucle consta de tres partes:

- Condición
- Cuerpo del bucle
- Salida del bucle

4.2 Diseño de la Estructura de algoritmos repetitivos

Para construir algoritmos repetitivos se utilizan estructuras repetitivas cuando se quiere que un conjunto de instrucciones se ejecuten un cierto número finito de veces, por ejemplo, escribir algo en pantalla cierta cantidad de veces, mover un objeto de un punto a otra cierta cantidad de pasos, o hacer una operación matemática cierta cantidad de veces, las estructuras repetitivas nos permiten hacerlo de forma sencilla.

4.3 Contadores y Acumuladores

4.3.1. Contadores

Es una variable en la memoria que **se incrementará en un valor constante** cada vez que se ejecute el proceso.

El contador se utiliza para llevar la cuenta de determinadas acciones que se pueden solicitar durante la resolución de un problema.

En las instrucciones de preparación se realiza la inicialización del contador o contadores. La inicialización consiste en poner el valor inicial de la variable que representa al contador. Generalmente se inicializa con el valor 0.

La forma básica general de un contador es:

$$\text{contador} = \text{contador} + \text{constante}$$

4.3.2 Acumulador

Un **acumulador** es una variable en la memoria cuya misión es almacenar cantidades variables.

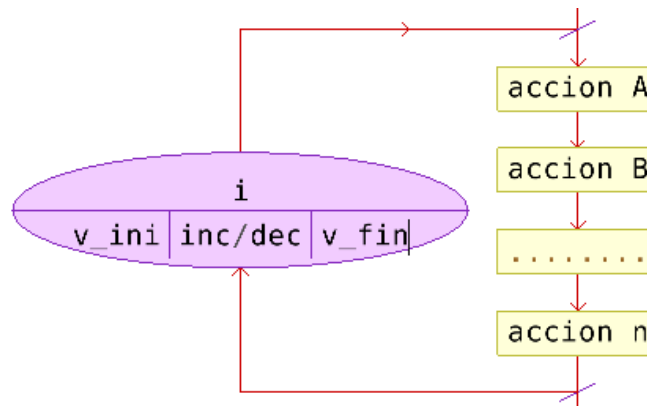
La forma general de un acumulador sería la siguiente:

$$\text{acumulador} = \text{acumulador} + \text{variable}$$

4.4 Estructura Para (for)

Esta estructura ejecuta las acciones del cuerpo del bucle un número determinado de veces, y de modo automático controla el número de iteraciones o pasos.

En un diagrama de flujo se puede apreciar de la siguiente manera:



En pseudocódigo:

```

Para i<-v_ini Hasta v_fin Con Paso inc/dec Hacer
    accion A;
    accion B;
    .....
    accion n;
FinPara

```

En el lenguaje C++ (código):

```

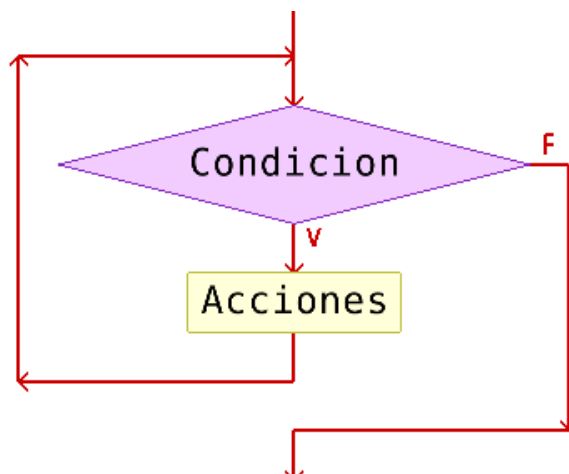
for ( inicio; condicion; incremento ) {
    sentencia(s);
}

```

4.5 Estructura Mientras – hacer (While)

Repite las acciones mientras la condición es verdadera. Cuando la condición es falsa sale del mientras.

En un diagrama de flujo se puede apreciar de la siguiente manera:



En pseudocódigo:

Mientras (expresion_logica) **hacer**

acción 1

acción 2

acción 3

....

acción n

Fin Mientras

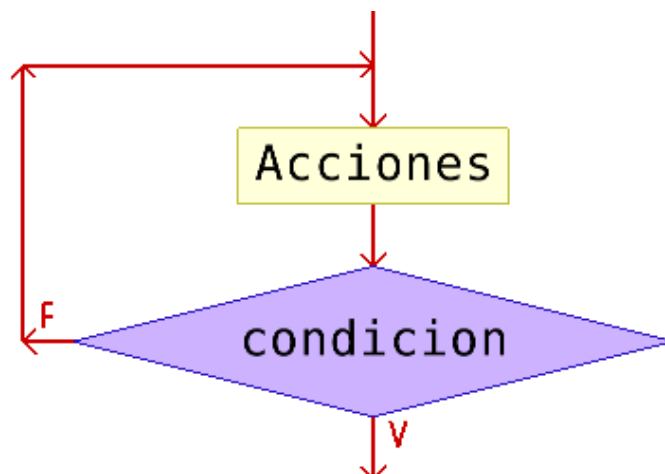
En el lenguaje C++ (código)

```
while(condicion)
{
    sentencias(s);
}
```

4.6 Estructura Repetir – Hasta (Do - While)

La estructura REPETIR HASTA cumple la misma función que la estructura MIENTRAS. La diferencia está en que la estructura MIENTRAS comprueba la condición al inicio y repetir lo hace al final. Es por ello que la estructura REPETIR HASTA se ejecuta por lo menos una vez.

En un diagrama de flujo se puede apreciar de la siguiente manera:



En pseudocódigo

Repetir

acción 1
acción 2
acción 3
....
acción n

Hasta (condición)

En el lenguaje C++ (código)

```
do {  
    sentencias(s);  
}  
while( condicion );
```

4.7 Estructuras repetitivas anidadas

Un ciclo anidado no es más que uno o más ciclos dentro de otro y de hecho no tenemos límite alguno para la cantidad de ciclos anidados.

Uno de los mayores problemas en cuanto a eficiencia que hay en la programación es tener ciclos anidados, son simplemente terribles para el rendimiento, sin embargo, hay ocasiones en las que son indispensables, entre más ciclos se tenga, uno dentro de otro, más lenta será la ejecución.

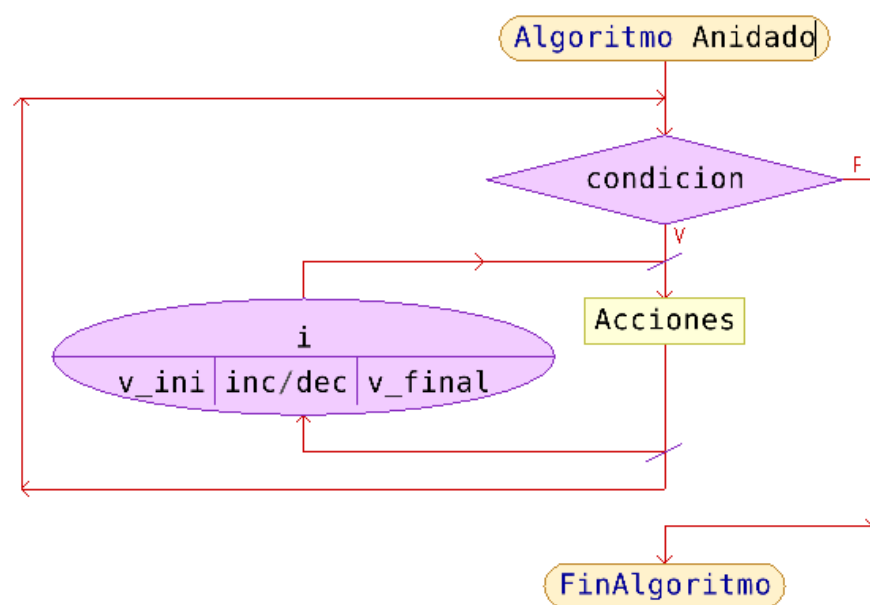
Ejemplo en Seudocódigo:

```

estructura anidado.psc x
1  Algoritmo Anidado
2
3  Mientras expresion logica Hacer
4
5      Para variable numerica<-valor inicial Hasta valor final Con Paso paso Hacer
6
7          secuencia de acciones
8
9      Fin Para
10
11  Fin Mientras
12
13  FinAlgoritmo
14
15

```

Ejemplo en Diagrama de flujo:



4.8 Aplicaciones al algebra y cálculo

1.- Hacer un programa que cuente del uno al 10


```
contardel 1 al10.psc x
1  Algoritmo Contar
2      definir x como entero;
3      x<-1 ;
4
5      Mientras x<=10 hacer
6          |
7              Escribir x ;
8
9              x= x + 1 ;
10         |
11     Fin Mientras
12
13 FinAlgoritmo
14
```

2.- Hacer un programa que cuente del 20 a 1 en forma descendente

```
contar del 20 al 1.psc x
1  Algoritmo Contar_descendente
2      definir x como entero;
3      x= 20;
4
5      Mientras x>=1 hacer
6          |
7              Escribir x ;
8              x = x - 1
9          |
10     Fin Mientras
11
12 FinAlgoritmo
```

3. Dado a, b determinar el valor de la división entera y el resto de la división entera de a, b sin usar los operadores de DIV y MOD

```
resto_divi.psc x
1  Algoritmo entero_resto
2      definir a, b, cv, pe, resto como entero;
3      Escribir "Ingrese valores de a y b";
4      Leer a, b
5      cv = 0;
6      mientras a >= b hacer
7          a = a - b;
8          cv = cv + 1;
9      fin mientras
10     pe = cv;
11     resto = a;
12     Escribir "La division entera es = ", pe;
13     Escribir "El resto es = ", resto;
14 FinAlgoritmo
15
```

4. Mostrar los números enteros menores a 20, comenzando desde 10 utilizando do – while en C++

```
#include <iostream>
using namespace std;

int main () {
    // Local variable declaration:
    int a = 10;

    // do loop execution
    do {
        cout << "value of a: " << a << endl;
        a = a + 1;
    } while( a < 20 );

    return 0;
}
```

5. Mostrar los números enteros menores a 20, comenzando desde 10 utilizando for en C++

```
#include <iostream>
using namespace std;

int main () {
    // for loop execution
    for( int a = 10; a < 20; a = a + 1 ) {
```

```
        cout << "value of a: " << a << endl;  
    }  
  
    return 0;  
}
```

4.9 Ejercicios Propuestos

Resolver cada uno de los siguientes ejercicios utilizando las tres sentencias repetitivas del tema.

- 1.- Hacer un programa que sume los 10 pares que le siguen al 24
- 2.- Hacer un programa que sume los 10 impares que le siguen al número n
- 3.- Hacer un programa que sume los pares comprendidos entre 100 y 200 sin considerar el rango de números comprendidos entre 50 y 76
- 5.- Hacer un programa que sume los 10 primeros números pares, luego que sume los 10 primeros Impares y muestre la diferencia de ambos resultados.

V. FUNDAMENTOS DE PROGRAMACION.

Definición

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático.

Un lenguaje de programación permite a un programador especificar de *manera precisa*: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar *relativamente* próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico.

Un programa escrito en un lenguaje de programación necesita pasar por un proceso de compilación, es decir, ser traducido al lenguaje de máquina, o ser interpretado para que pueda ser ejecutado por el ordenador. También existen lenguajes de scripting que son ejecutados a través de un intérprete y no necesitan compilación.

5.1 Proceso de codificación, compilación y ejecución de un programa

b) Codificación

Codificación es el proceso de conversión en símbolos de una determinada información con el fin de ser comunicada, y a efectos de ser entendida por el receptor, aplicando las reglas de un código predeterminado. Es decir que en la codificación el emisor convierte sus ideas en signos que sean fácilmente comprendidos por quienes reciben la información.

c) Compilación

Proceso de traducción de un código fuente (escrito en un lenguaje de programación de alto nivel) a lenguaje máquina (código objeto) para que pueda ser ejecutado por la computadora. Las computadoras sólo entienden el lenguaje máquina. La aplicación o la herramienta encargada de la traducción se llama compilador.

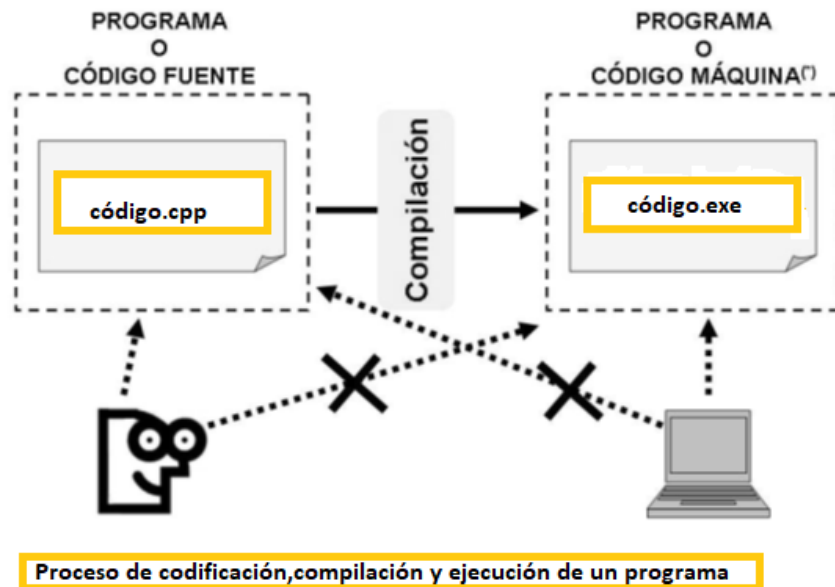
d) Ejecución

En informática, ejecutar es la acción de iniciar la carga de un programa o de cualquier archivo ejecutable.

En otras palabras, la ejecución es el proceso mediante el cual una computadora lleva a cabo las instrucciones de un programa informático.

Se pueden ejecutar programas compilados (por ejemplo, en Windows, los .EXE) o programas interpretados (por ejemplo, los scripts).

Ejecutar un programa implica que éste estará en estado de ejecución y, por ende, en memoria, hasta que se finalice.

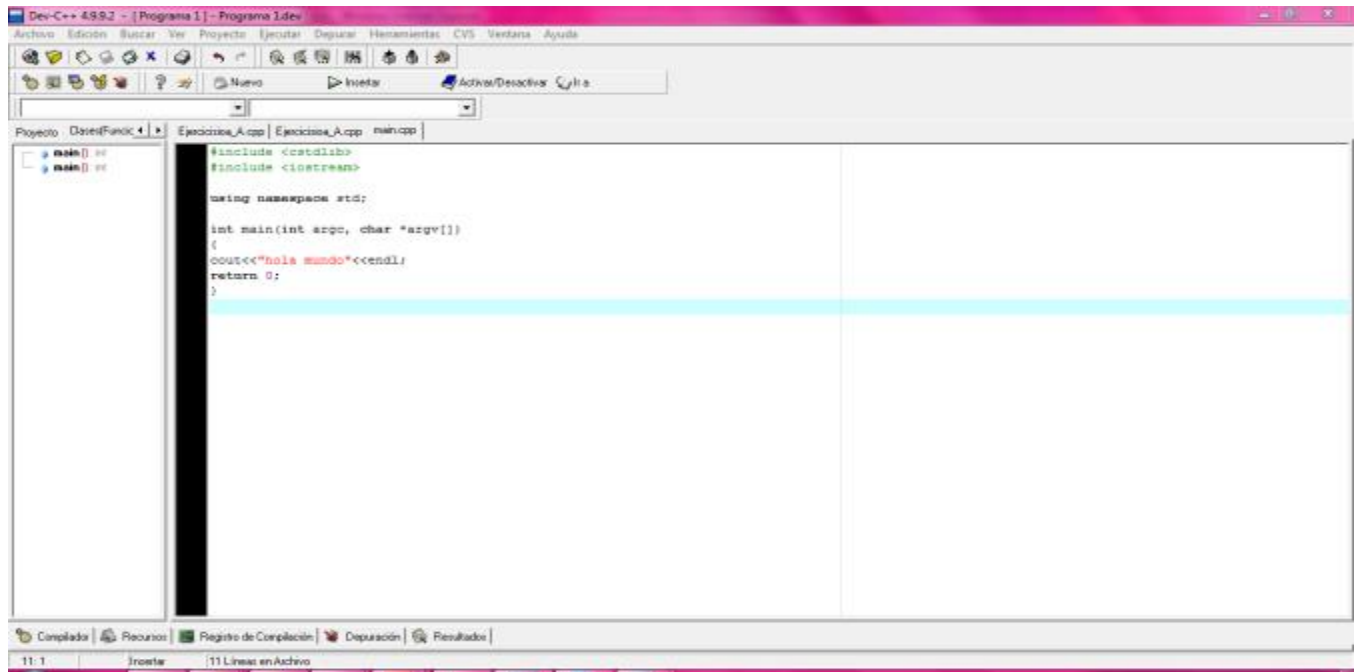


El proceso de ejecución de un programa escrito en un lenguaje de programación y mediante un compilador tiene los siguientes pasos:

- Escritura del programa fuente en un editor (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento.
- Introducir el programa fuente en memoria.
- Compilar el programa con el compilador.
- Verificar y corregir errores de compilación.
- Obtención del programa objeto
- El enlazador (linker) obtiene el programa ejecutable.
- Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.

5.2 Estructura de un programa

La estructura de un programa en dev c++ es como se muestra a continuación



`#include <iostream>`

declaración de librerías

`int main(){`

función main

`cout<<"hola mundo"<<endl;`

secuencia de instrucciones

`return 0;`

valor de retorno de la función

`}`

llaves de cierre de la función

#include <iostream>

La parte del **#include** se refiere a la biblioteca de funciones que va a utilizar. Es decir, para llamar a una biblioteca en particular debemos hacer lo siguiente:

#include <librería_solicitada>

El estándar de C++ incluye varias bibliotecas de funciones, y dependiendo del compilador que se esté usando, puede aumentar el número.

```
int main()
{
...
}
```

Todo programa en C++ comienza con una función **main()**, y sólo puede haber una.

En c++ el **main()** siempre regresa un entero, es por eso se antepone “**int**” a la palabra “**main**”. La llave que se abre significa que se iniciará un bloque de instrucciones.

```
cout<< “hola mundo”<<endl;
```

Esta es una instrucción. La instrucción cout está definida dentro de la biblioteca **iostream**, que previamente declaramos que íbamos a utilizar. Una función, en este caso **main()** siempre comienza su ejecución con una instrucción (la que se encuentra en la parte superior), y continúa así hasta que se llegue a la última instrucción (de la parte inferior). Para terminar una instrucción siempre se coloca “;”.

```
return 0;
```

Esta es otra instrucción, en este caso la instrucción return determina que es lo que se devolverá de la función **main()**. Habíamos declarado que **main** devolvería un entero, así que la instrucción **return** devuelve 0. Lo cual a su vez significa que no han ocurrido errores durante su ejecución.

```
}
```

La llave de cierre de la función main() indica el término del bloque de instrucciones.

En algunos programas de ejemplo, notarán el uso de dobles diagonales (“//”). Estas diagonales se usan para escribir comentarios de una línea dentro del código del programa. Además, podrá encontrar el uso de “/*” “*/” estos caracteres encierran un comentario de varias líneas y cualquier cosa que se escriba dentro de ella no influenciará en el desempeño del programa.

5.3 Funciones proporcionadas por el compilador

El lenguaje dev C++, como la mayoría de los lenguajes de programación, permite el uso de “bibliotecas” con funciones predefinidas que se pueden utilizar en cualquier programa. Uso de Funciones Predefinidas.

Se utilizará la función sqrt (square root = raíz cuadrada) para ejemplificar el uso de funciones predefinidas.

Algunas Funciones Predefinidas Algunas funciones predefinidas se describen en la Tabla siguiente:

Nombre	Descripción	Tipo de Argumentos	Tipo de Valor de Regreso	Ejemplo	Valor	Biblioteca
sqrt	Raíz Cuadrada	double	double	sqrt(4.0)	2.0	math.h
pow	Potencia	double	double	pow(2.0,3.0)	8.0	math.h
abs	Valor absoluto de un int	int	int	abs(-7) abs(7)	7	stdlib.h
fabs	Valor absoluto de un double	double	double	fabs(-7.5) fabs(7.5)	7.5	math.h
ceil	Redondeo hacia el número inmediato superior	double	double	ceil(3.2) ceil(3.9)	4.0	math.h
floor	Redondeo hacia el número inmediato inferior	double	double	floor(3.2) floor(3.9)	3.0	math.h
sin	Seno	double	double	sin(0.0)	0.0	math.h
cos	Coseno	double	double	cos(0.0)	1.0	math.h
tan	Tangente	double	double	tan(0.0)	0.0	math.h

La más complicada de las funciones de la tabla es la función pow que sirve para obtener la potencia de un número. Por ejemplo, las siguientes sentencias son un ejemplo de aplicación de la función pow:

```
double resultado, x=3.0, y=2.0;
```



```
resultado = pow(x,y);  
cout<< resultado;
```

Las sentencias anteriores mostraran en pantalla al número 9.0.

Ejemplo:

El siguiente programa calcula las raíces la ecuación cuadrática $ax^2 + bx + c = 0$

```
#include <iostream.h>  
#include <stdlib.h>  
#include <math.h>  
  
int main()  
{  
    /* Este programa permite el calculo de las raices  
       de una ecuacion cuadratica */  
  
    /* Declaración de variables*/  
    double a, b, c, x_1, x_2;  
  
    /* Entrada de datos */  
    cout<< "Dame los coeficientes a,b y c de la ecuacion cuadratica \n";  
    cin>>a >> b >> c;  
  
    /* Procesamiento de datos */  
  
    x_1 = ( -b + sqrt( pow(b,2.0) - 4.0 * a * c) ) / (2.0 * a);  
    x_2 = ( -b - sqrt( pow(b,2.0) - 4.0 * a * c) ) / (2.0 * a);  
  
    /* Salida de Resultados */
```

5.4 Aplicaciones empleando estructuras secuenciales, selectivas y repetitivas.

```
// Programa para sentencia SWITCH CASE  
#include<stdio.h>  
#include<iostream.h>  
#include<conio.h>  
#include<stdlib.h>  
// Programa que calcula la venta de helados hasta que el usuario desea parar  
int main()  
{  
    char opcion;  
    int c1=0,c2=0,c3=0,c4=0,c5=0,sum1=0,sum2=0,sum3=0,sum4=0,sum5=0,res=1;  
    while (res != 0)// ciclo que controla la salida del programa  
    {  
        // Menu que permitira elegir al usuario un producto
```

```

cout<<"introduzca el sabor de su preferencia \n";
cout<<" FRUTILLA    --> F \n";
cout<<" CHOCOLATE   --> C \n";
cout<<" VAINILLA    --> V \n";
cout<<" CHIRIMOYA   --> Y \n";
cout<<" PASAS AL RON --> R \n";
cin>> opcion; // lectura de la demanda del usuario
//Estructura de selección de operacion de acuerdo a la opción del usuario
switch (opcion)
{case 'F': c1=c1+1;sum1=sum1+5;break; // Si el usuario eligio F de Frutilla
case 'C': c2=c2+1;sum2=sum2+6;break; // Si el usuario eligio C de Chocolate
case 'V': c3=c3+1;sum3=sum3+5;break;
case 'Y': c4=c4+1;sum5=sum5+5;break;
default: c5=c5+1;sum5=sum5+5; // En caso que no eligio ninguna de las anteriores
opciones.
}
// Impresion de resultados
cout<<"se vendieron  "<<c1<<" helados de frutilla con  "<< sum1 << " de ganancia \n";
cout<<"se vendieron  "<<c2<<" helados de chocolate con  "<< sum2 << " de ganancia
\n";
cout<<"se vendieron  "<<c3<<" helados de vainilla con  "<< sum3 << " de ganancia
\n";
cout<<"se vendieron  "<<c4<<" helados de chirimoya con  "<< sum4 << " de ganancia
\n";
cout<<"se vendieron  "<<c5<<" helados de pasas al ron con  "<< sum5 << " de ganancia
\n";
cout<<"desea continuar?  0-->NO  1--> SI \n";
cin >> res; // variable que controla la salida del programa
}
getch();
return 0;
}

```

5.5 Ejercicios Propuestos

1) Una persona debe realizar un muestreo con 50 personas para determinar el promedio de peso de los niños, jóvenes, adultos y viejos que existen en su zona habitacional. Se determinan las categorías con base en la siguiente tabla:

CATEGORIA	EDAD
Niños	0 - 12
Jóvenes	13 - 29
Adultos	30 - 59
Viejos	60 en adelante

Al cerrar un expendio de naranjas, 15 clientes que aún no han pagado recibirán un 15% de descuento si compran más de 10 kilos. Determinar cuánto pagará cada cliente y cuanto percibirá la tienda por esas compras.

3) En un centro de verificación de automóviles se desea saber el promedio de puntos contaminantes de los primeros 25 automóviles que lleguen. Asimismo, se desea saber los puntos contaminantes del carro que menos contamina y del que más contamina.

4) Un entrenador le ha propuesto a un atleta recorrer una ruta de cinco kilómetros durante 10 días, para determinar si es apto para la prueba de 5 Kilómetros o debe buscar otra especialidad. Para considerarlo apto debe cumplir por lo menos una de las siguientes condiciones:

- Que en ninguna de las pruebas haga un tiempo mayor a 16 minutos.
- Que al menos en una de las pruebas realice un tiempo mayor a 16 minutos.
- Que su promedio de tiempos sea menor o igual a 15 minutos.

5) Obtener el n -ésimo término de la serie Fibonacci.

VI. ARREGLOS

Definición

Un array o arreglo es un conjunto de datos que se almacenan en memoria de manera seguida con el mismo nombre. Es una colección de datos del mismo tipo, cada una de ellas se llama elemento y posee una posición dentro del arreglo llamado índice que se incrementa de uno en uno.

Se clasifican:

- Unidimensionales
- Bidimensionales

6.1 Arreglos Unidimensionales

Un arreglo unidimensional llamado también vector maneja un solo índice que indica la posición de un elemento del vector. La primera posición del array es la posición 0.

Podríamos agrupar en un array una serie de elementos de tipo enteros, flotantes, caracteres, objetos, etc.

Crear un vector en C++ es sencillo, seguimos la siguiente sintaxis:

Tipo nombre[tamaño_Max];

Ejemplo: `int A[30];`

Ejemplo 1: Realizar un algoritmo para almacenar números enteros y mostrar dichos números.

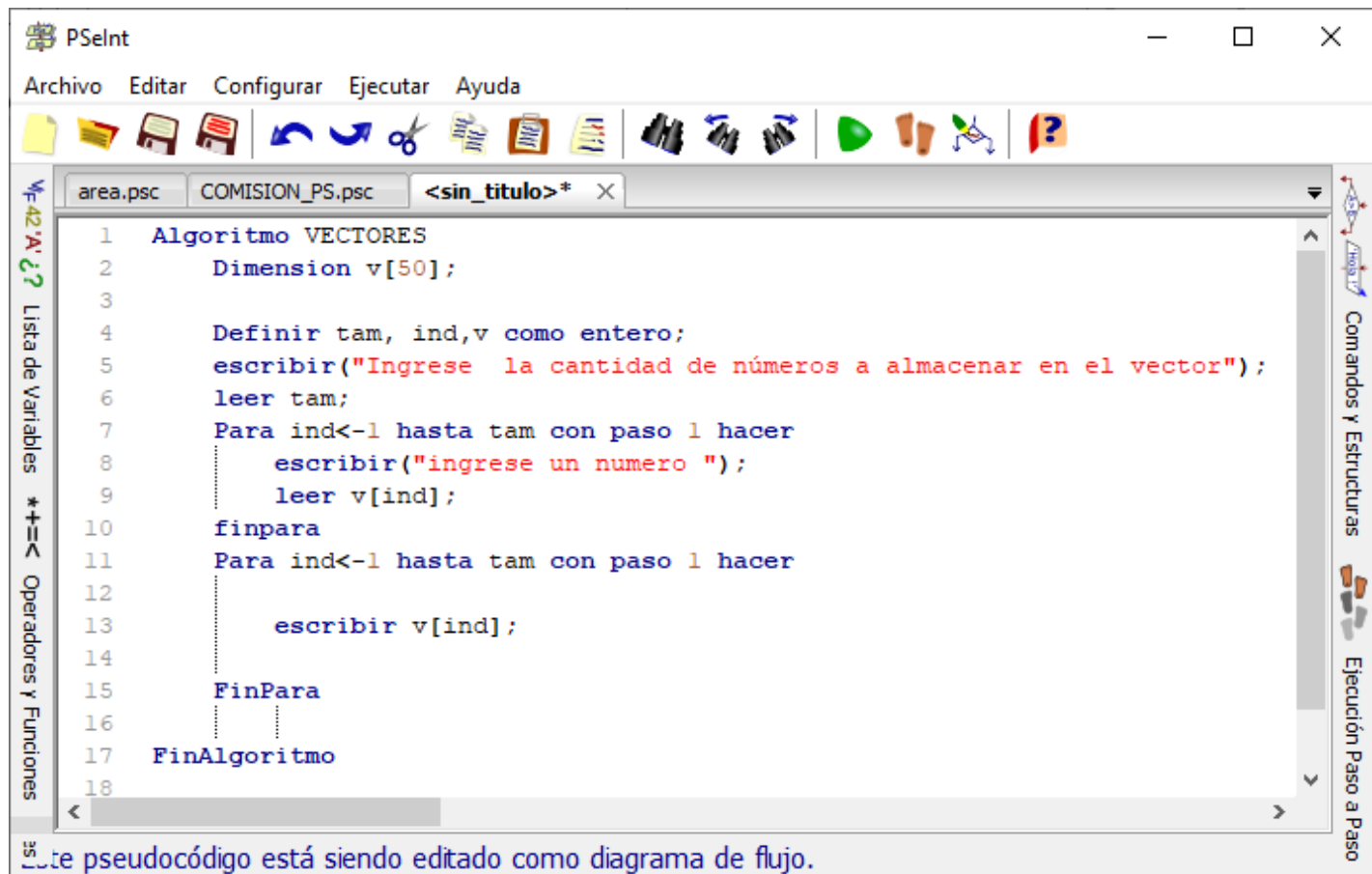
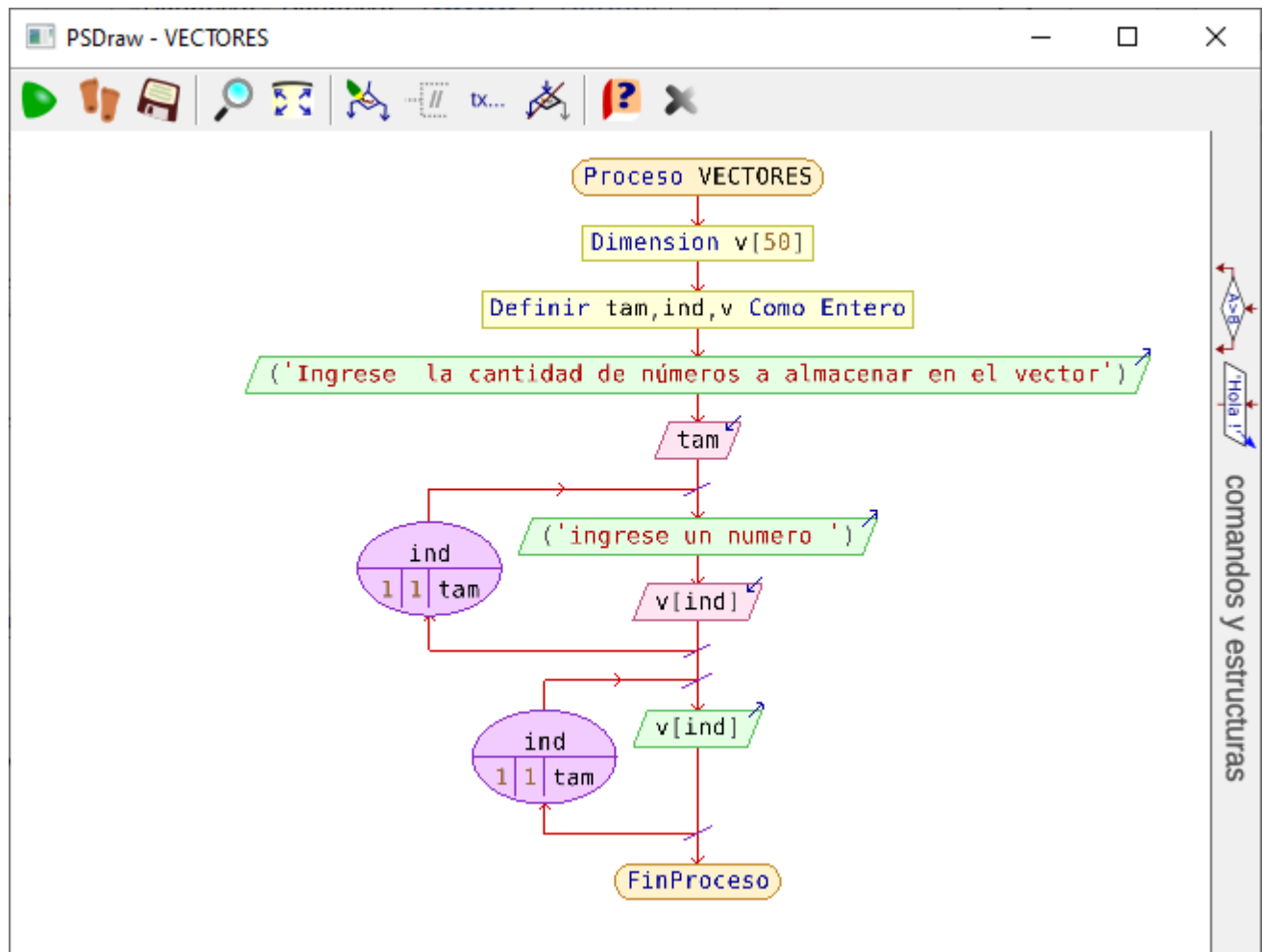
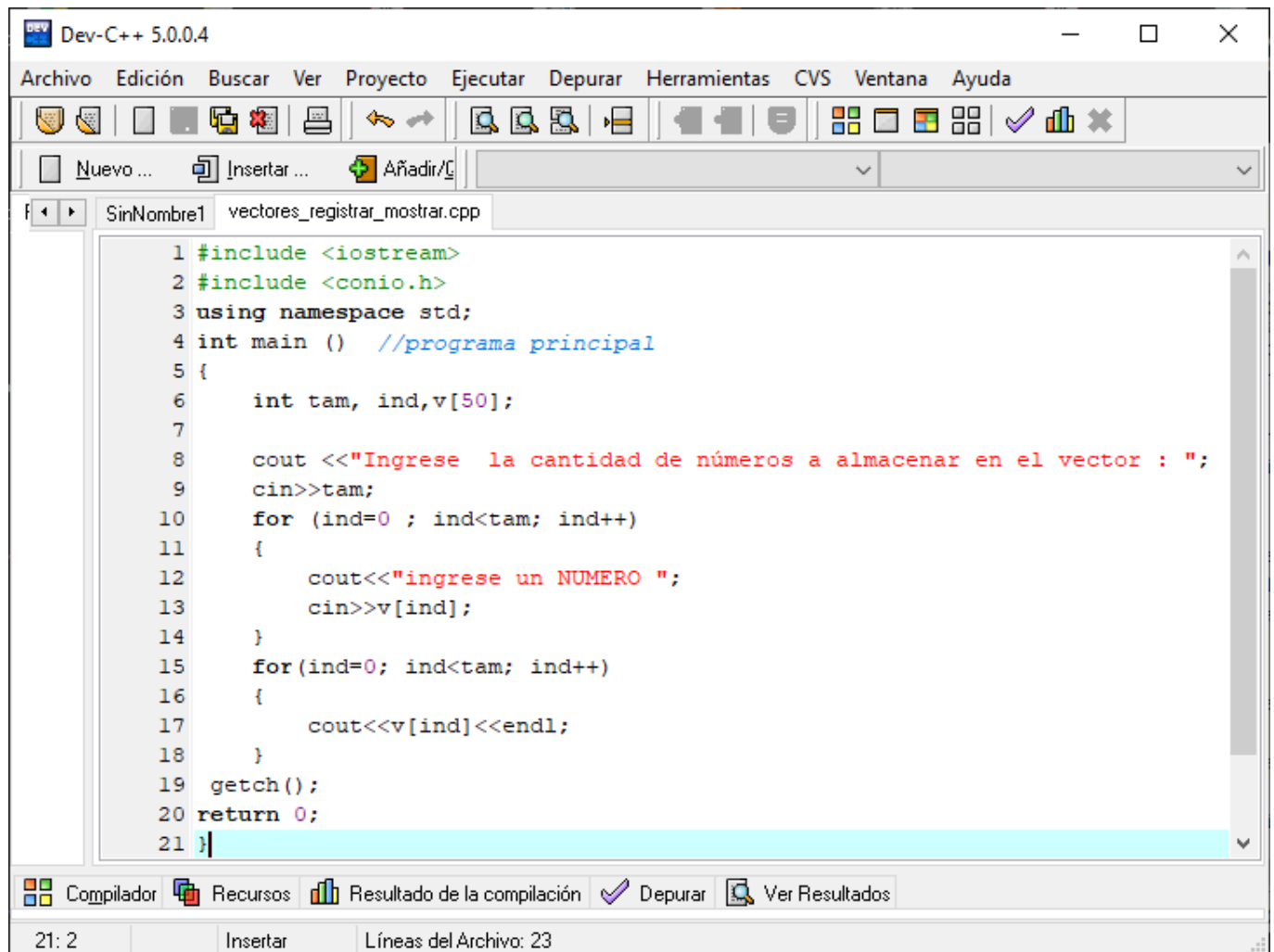


DIAGRAMA DE FLUJO



LENGUAJE DE PROGRAMACIÓN C++

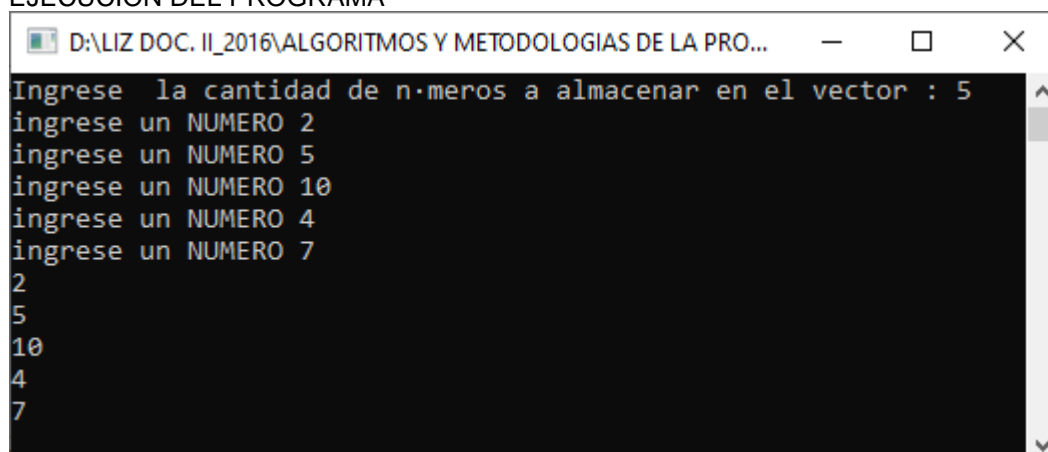


The screenshot shows the Dev-C++ 5.0.0.4 IDE. The menu bar includes Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Depurar, Herramientas, CVS, Ventana, and Ayuda. The toolbar contains icons for file operations and execution. The file explorer on the left shows a project named 'SinNombre1' with a file 'vectores_registrar_mostrar.cpp'. The main editor displays the following C++ code:

```
1 #include <iostream>
2 #include <conio.h>
3 using namespace std;
4 int main () //programa principal
5 {
6     int tam, ind, v[50];
7
8     cout << "Ingrese la cantidad de números a almacenar en el vector : ";
9     cin >> tam;
10    for (ind=0 ; ind<tam; ind++)
11    {
12        cout << "ingrese un NUMERO ";
13        cin >> v[ind];
14    }
15    for (ind=0; ind<tam; ind++)
16    {
17        cout << v[ind] << endl;
18    }
19    getch();
20    return 0;
21 }
```

The status bar at the bottom shows '21: 2', 'Insertar', and 'Líneas del Archivo: 23'.

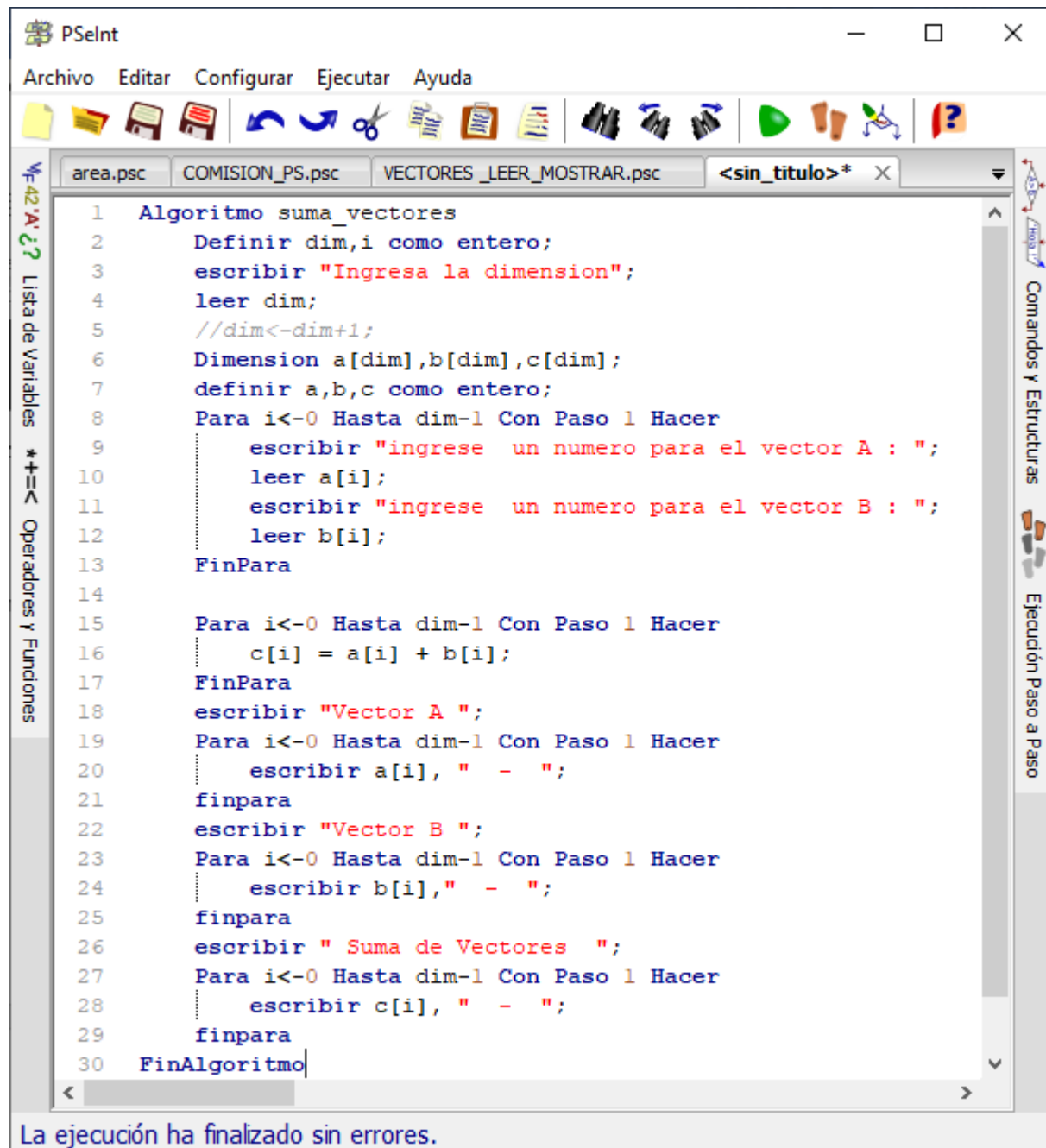
EJECUCIÓN DEL PROGRAMA



The screenshot shows a command prompt window with the title 'D:\LIZ DOC. II_2016\ALGORITMOS Y METODOLOGIAS DE LA PRO...'. The output of the program is as follows:

```
Ingrese la cantidad de n-meros a almacenar en el vector : 5
ingrese un NUMERO 2
ingrese un NUMERO 5
ingrese un NUMERO 10
ingrese un NUMERO 4
ingrese un NUMERO 7
2
5
10
4
7
```

Ejemplo 2 : Realizar algoritmo y un programa en C++ para sumar 2 vectores a y b y poner el resultado en un tercer vector c:



```
1  Algoritmo suma_vectores
2      Definir dim,i como entero;
3      escribir "Ingresa la dimension";
4      leer dim;
5      //dim<-dim+1;
6      Dimension a[dim],b[dim],c[dim];
7      definir a,b,c como entero;
8      Para i<-0 Hasta dim-1 Con Paso 1 Hacer
9          escribir "ingrese un numero para el vector A : ";
10         leer a[i];
11         escribir "ingrese un numero para el vector B : ";
12         leer b[i];
13     FinPara
14
15     Para i<-0 Hasta dim-1 Con Paso 1 Hacer
16         c[i] = a[i] + b[i];
17     FinPara
18     escribir "Vector A ";
19     Para i<-0 Hasta dim-1 Con Paso 1 Hacer
20         escribir a[i], " - ";
21     finpara
22     escribir "Vector B ";
23     Para i<-0 Hasta dim-1 Con Paso 1 Hacer
24         escribir b[i], " - ";
25     finpara
26     escribir " Suma de Vectores ";
27     Para i<-0 Hasta dim-1 Con Paso 1 Hacer
28         escribir c[i], " - ";
29     finpara
30     FinAlgoritmo
```

La ejecución ha finalizado sin errores.

El algoritmo desarrollado en C++


```

1 #include <iostream>
2 #include <conio.h>
3 using namespace std;
4 int main()
5 {   int dim;
6     cout << "Ingresa la dimension" << endl;
7     cin >> dim;
8     int a[dim];
9     int b[dim];
10    int c[dim];
11
12    for(int i = 0; i < dim; i++) {
13        cout<<"ingrese un numero para el vector A : ";
14        cin>>a[i];
15        cout<<"ingrese un numero para el vector B : ";
16        cin>>b[i];
17    }
18    for (int i = 0; i < dim; i++) {
19        c[i] = a[i] + b[i];
20    }
21    cout << "Vector A " << endl;
22    for(int i = 0; i < dim; i++)
23    {
24        cout << a[i] << " - ";
25    }
26
27    cout << endl << endl;
28    cout << "Vector B " << endl;
29    for(int i = 0; i < dim; i++)
30    {
31        cout << b[i] << " - ";
32    }
33    cout << endl << endl;
34    cout << " Suma de Vectores " << endl;
35    for(int i = 0; i < dim; i++)
36    {
37        cout << c[i] << " - ";
38    }
39    cout << endl << endl;
40    getch();
41 }
42

```

6.2 Ordenación y Búsqueda

6.2.1 Ordenación

Debido a que las estructuras de datos son utilizadas para almacenar información, para poder recuperar esa información de manera eficiente es deseable que aquella esté ordenada.

Existen varios métodos para ordenar las diferentes estructuras de datos básicas.

En general los métodos de ordenamiento no son utilizados con frecuencia, en algunos casos sólo una vez.

Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento.

Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

```
#include<stdio.h>
#include<conio.h>
int a[9]={3,5,2,1,6,8,7,4};
int i,j,aux,n=9;
int main(){
clrscr();
for(i=0;i<n;i++)

    {
        for(j=0;j<n-1;j++)

            {
                if(a[j]>a[j+1]){
                    aux=a[j];
                    a[j]=a[j+1];
                    aux=a[j];
```

```

        a[j+1]=aux;
    }
}

return 0;

}

```

6.2.2 Búsqueda

La búsqueda es una operación que tiene por objeto la localización de un elemento dentro de la estructura de datos. A menudo un programador estará trabajando con grandes cantidades de datos almacenados en arreglos y pudiera resultar necesario determinar si un arreglo contiene un valor que coincide con algún valor clave o buscado.

Siendo el array de una dimensión o lista una estructura de acceso directo y a su vez de acceso secuencial, encontramos dos técnicas que utilizan estos dos métodos de acceso, para encontrar elementos dentro de un array: búsqueda lineal y búsqueda binaria.

Búsqueda Secuencial:

La búsqueda secuencial es la técnica más simple para buscar un elemento en un arreglo. Consiste en recorrer el arreglo elemento a elemento e ir comparando con el valor buscado (clave). Se empieza con la primera casilla del arreglo y se observa una casilla tras otra hasta que se encuentra el elemento buscado o se han visto todas las casillas. El resultado de la búsqueda es un solo valor, y será la posición del elemento buscado o cero. Dado que el arreglo no está en ningún orden en particular, existe la misma probabilidad de que el valor se encuentra ya sea en el primer elemento, como en el último. Por lo tanto, en promedio, el programa tendrá que comparar el valor buscado con la mitad de los elementos del arreglo.

El método de búsqueda lineal funciona bien con arreglos pequeños o para arreglos no ordenados. Si el arreglo está ordenado, se puede utilizar la técnica de alta velocidad de búsqueda binaria, donde se reduce sucesivamente la operación eliminando repetidas veces la mitad de la lista restante.

Código en dev C++ de búsqueda secuencial

```

int i;

bool band=false;

for(i=0; i<n; i++){

    if(dato==vector[i]) {

        band=true;

        cout<< "el dato existe";

        }

    }

    if(band==false)

        cout<< "el dato no existe en el vector";

```

6.3 Arreglos Bidimensionales

A los arreglos de dos dimensiones (bidimensionales) se les conoce como matrices y son estructuras de datos que organizan su información en forma de tablas, es decir, los elementos que la conforman están dispuestos bajo dos conceptos de clasificación (fila y columna). Para poder indicar el lugar donde se encuentra un determinado elemento, es necesario utilizar dos índices: uno para indicar el renglón o fila y otro para indicar la columna. Puede mirarse una matriz como un vector de vectores; por lo tanto, es un conjunto de componentes en el que se necesitan dos subíndices para identificar un elemento que pertenezca al arreglo.

Gráficamente podemos representar al arreglo bidimensional como se muestra a continuación:

	0	1	2	3	4	5
0						
1						
2						

Un arreglo bidimensional $N * M$ tiene N filas y M columnas; por lo tanto, tiene $N * M$ elementos dispuestos interiormente en memoria en forma sucesiva

En Pseint se declara una matriz:

```
Dimension matriz[10,15];
```

Una matriz se declara en c++ :

```
Tipo Nombre_matriz[ tam_max_filas][tam_max_cols]
```

Ejemplo : `int matriz[10][15];`

Ejemplo 1: El presente ejemplo muestra como almacenar elementos dentro de las posiciones de un arreglo de dos dimensiones.

PSEUDOCÓDIGO EN PSEINT

```

Proceso Almacernar_matriz
  Definir matriz, filas, cols, f, c Como Entero;
  filas <- 3;
  cols <- 4;
  Dimension matriz[filas, cols]; //arreglo de dos dimensiones (12 elementos)
  Para f <- 0 Hasta filas-1 Con Paso 1 Hacer
    Para c <- 0 Hasta cols-1 Con Paso 1 Hacer
      Escribir 'Ingrese el elemento [' , f, ', ', c, ']' ;
      Leer matriz[f, c]; //se almacena el valor
    FinPara
  FinPara
  Escribir ""; //salto de linea
  Escribir ""; //salto de linea

  //Visualizar la tabla de valores de forma matricial
  Escribir "*****TABLA DE VALORES*****";
  Escribir ""; //salto de linea
  Para f <- 0 Hasta filas-1 Con Paso 1 Hacer
    Para c <- 1 Hasta cols-1 Con Paso 1 Hacer
      Escribir Sin Saltar " ", matriz[f, c]; //se visualiza el valor
    FinPara
    Escribir ""; //salto de linea
  FinPara
  Escribir ""; //salto de linea
  Escribir ""; //salto de linea

FinProceso

```

En C++

```

1 #include <iostream>
2 #include <conio.h>
3 using namespace std;
4 int main()
5 {
6     int filas , cols, f,c;
7     filas=3;
8     cols=4;
9     int matriz[filas][cols];
10
11     for( f = 0; f <= filas-1; f++)
12     {
13         for( c = 0; c <= cols-1; c++)
14         {
15             cout<<"ingrese el elemento [ " <<f <<" , "<<c<<" ]= ";
16             cin>>matriz[f][c];
17         }
18     }
19     cout<<endl;
20     cout<<endl;
21     cout<<" *****TABLA DE VALORES *****"<<endl;
22     for( f = 0; f <= filas-1; f++)
23     {
24         for( c = 0; c <= cols-1; c++)
25         {
26             cout<< matriz[f][c]<<"    ";
27         }
28         cout<<endl;
29     }
30     getch();
31 }

```

EJECUCIÓN DEL PROGRAMA

```
D:\LIZ DOC. II_2016\ALGORI...
ingrese el elemento [ 0, 0 ]= 2
ingrese el elemento [ 0, 1 ]= 3
ingrese el elemento [ 0, 2 ]= 4
ingrese el elemento [ 0, 3 ]= 5
ingrese el elemento [ 1, 0 ]= 6
ingrese el elemento [ 1, 1 ]= 2
ingrese el elemento [ 1, 2 ]= 4
ingrese el elemento [ 1, 3 ]= 1
ingrese el elemento [ 2, 0 ]= 6
ingrese el elemento [ 2, 1 ]= 7
ingrese el elemento [ 2, 2 ]= 2
ingrese el elemento [ 2, 3 ]= 6

*****TABLA DE VALORES *****
2    3    4    5
6    2    4    1
6    7    2    6
```

6.4 Ejercicios Propuestos

1. Llenar un vector con las ventas que se realizaron durante una semana. Mostrar la venta mayor y el día. Considerar que puede haber varios días con la venta mayor.
2. Llenar un vector con n notas y ordenar en forma ascendente, luego insertar una nota en la posición que le corresponde.
3. Visualizar las siguientes matrices: La dimensión de la matriz en $n * m$

1 2 3 4	1	1 3 5
2 4 6 8	1 2	2 4 6
3 6 9 12	1 2 3	7 9 11
4 8 12 16	1 2 3 4	8 10 12
4. Realizar la multiplicación de 2 matrices.
5. Hacer un algoritmo que encuentre e imprima la matriz transpuesta de una matriz MAT. La matriz transpuesta de la matriz MAT se encuentra intercambiando las filas por las columnas y las columnas por las filas. Si T_{MAT} es la matriz transpuesta de MAT, implica entonces que T_{MAT} [columna, fila] es igual a MAT [fila, columna]. Si el contenido de MAT es:

MAT=

5	8	1	9
15	11	9	4
16	4	3	0

TMAT=

5	15	16
8	11	4
1	9	3
9	4	0

Como se puede ver, se invierte el orden de la matriz; es decir, el número de filas de MAT es igual al número de columnas de TMAT y el número de columnas se invierte por el número de filas de TMAT.

6. Realizar la multiplicación de 2 matrices. Verificar si tienen los tamaños adecuados.
7. Leer una matriz y obtener su opuesta por el método de Gauss Jordan.

VII. SUB PROGRAMAS

Definición

Uno de los métodos más conocidos para resolver un problema es dividir en problemas más pequeños, llamados subproblemas. De esta manera, en lugar de resolver una tarea compleja y tediosa, resolvemos otras más sencillas y a partir de ellas llegamos a la solución a esta técnica se le suele llamar diseño descendente, metodología del divide y vencerás o programación top-down.

A estos subprogramas se les suele llamar módulos, de ahí viene el nombre de programación modular.

En C++ consta al menos de una función **main** (programa principal) y otras funciones alternativas del programa. La ejecución de un programa comienza por la función main, cuando se llama a una función, el control se pasa a la misma para su ejecución y cuando finaliza el control es devuelto de nuevo al módulo que llamo para continuar con la ejecución del mismo a partir de la sentencia que efectuó la llamada.

7.1 PROCEDIMIENTO

Un procedimiento es un subprograma que realiza una tarea específica que proporciona cero, uno o varios valores en función de los parámetros definidos en su formato.

En el contexto de C++ un procedimiento es básicamente una función void que no nos obliga a utilizar una sentencia return.

Para invocarlo y hacer que se ejecute, basta con escribir su nombre en el cuerpo de otro modulo o en el programa principal anotando también los parámetros si es que existieran.

SINTAXIS

```
void nombre_proc (tipo p1, tipo p2,...)
{
    Sentencias ;
}
```

7.2 Funciones

Las funciones son un conjunto de procedimientos encapsulados en un bloque, usualmente reciben parámetros, cuyos valores utilizan para efectuar operaciones y adicionalmente retornan un valor. Esta definición proviene de la definición de función matemática la cual posee un dominio y un rango, es decir un conjunto de valores que puede tomar y un conjunto de valores que puede retornar luego de cualquier operación.

SINTAXIS

```
tipo nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento]...])  
{  
    Bloque de instrucciones  
    return valor;  
}
```

Recordemos que una función siempre retorna algo, por lo tanto, es obligatorio declararle un tipo (el primer componente de la sintaxis anterior), luego debemos darle un nombre a dicha función, para poder identificarla y llamarla durante la ejecución, después al interior de paréntesis, podemos poner los argumentos o parámetros. Luego de la definición de la "firma" de la función, se define su funcionamiento entre llaves; todo lo que esté dentro de las llaves es parte del cuerpo de la función y éste se ejecuta hasta llegar a la instrucción **return**.

Los argumentos o parámetros

Hay algunos detalles respecto a los argumentos de una función, veamos:

Una función o procedimiento pueden tener una cantidad cualquier de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más parámetros. Aunque habitualmente no suelen tener más de 4 o 5.

Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.

Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo de la función.

Consejos acerca de **return**

Debes tener en cuenta dos cosas importantes con la sentencia **return**:

Cualquier instrucción que se encuentre después de la ejecución de **return** NO será ejecutada. Es común encontrar funciones con múltiples sentencias **return** al interior de condicionales, pero una vez que el código ejecuta una sentencia **return** lo que haya de allí hacia abajo no se ejecutará.

El tipo del valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara **int**, el valor retornado debe ser un número entero.

7.3 APLICACIONES CON FUNCIONES Y PROCEDIMIENTOS

Ejemplo 1: Plantear un algoritmo que calcule el área de un triángulo, mediante un procedimiento y una función:

Procedimiento:

```
#include <stdio.h>
#include <stdlib.h>

void areatriangulo (void) {
    float base, altura;
    cout<<"Introduce base: ";
    cin>>base;
    cout<<"Introduce altura: ";
    cin>>altura;
    cout<<"El área es:"<<(base*altura)/2;
}

int main(int argc, char *argv[])
{
    areatriangulo();
    system("PAUSE");
    return 0;
}
```

Función:

```

#include <stdio.h>
#include <stdlib.h>
float areatriangulo (void) {
    float base, altura;
    cout<<"Introduce base: ";
    cin>>base;
    cout<<"Introduce altura: ";
    cin>>altura;
    return (base*altura)/2;
}
int main(int argc, char *argv[])
{
    float area;
    area=areatriangulo();
    cout<<"El área es: "<<area;
    system("PAUSE");
    return 0;
}

```

Ejemplo 2

```

#include <iostream.h>
#include <stdlib.h>
using namespace std;
// divide enteros
int divide(int a, int b)
{
    cout << "división entera" << endl;
    return ( (b != 0) ? a/b : 0);
}
// divide reales
double divide(double a, double b)
{
    cout << "división real" << endl;
    return ( (b != 0) ? a/b : 0);
}
// punto de prueba

```

```
int main()
{
    cout << divide(10, 3) << endl;
    cout << divide(10.0, 3.0) << endl;
    cin.get();
}
```

7.4 Ejercicios Propuestos

1. Realice un programa con funciones para calcular el mayor y menor de dos números.
2. Realice un programa con procedimientos para hallar la suma de dos números cualesquiera.
3. Calcular $\binom{n}{r} = \frac{n!}{r!(n-r)!}$ aplicando programación modular.
4. Calcular la suma de la siguiente serie:

$$s = \frac{1}{0!} + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^x}{n!}$$

Utilizar funciones o procedimientos para resolver el problema.

5. Realizar un procedimiento para ordenar un vector. Mostrar el vector desordenado y ordenado utilizar un procedimiento para mostrar dicho vector.
6. Llenar dos matrices con números enteros y determinar cuál de las dos matrices tiene la mayor suma de sus elementos. Utilizar un procedimiento para llenar la matriz y una función q devuelva la suma de sus elementos. Realizar el programa como se muestra a continuación.

```
void llenar ( int mat[ ][ ], int tf,int tc)
{

}

void mostrar(int mat[ ][ ], int tf,int tc)
{

}

int sumaElem(int mat[ ][ ], int tf,int tc)
{

return sum;
}
```

VIII. Bibliografía

- Luis Joyanes Aguilar, (Ed.). (2008). Fundamentos de Programación. Lima, Perú. Editorial Mc Graw Hill, 4º Edición.
- Koffman, Elliot B, (Ed.). (2004). Introducción al lenguaje y resolución de Problemas con programación. Mc Graw – Hill. Madrid, España.
- Wu, Thomas C, (Ed.). (2001). Programación en C++ algoritmos, estructuras de datos y objetos. México. Mc Graw – Hill, interamericana.
- Correa Uribe Guillermo, (Ed). (2003). Desarrollo de Algoritmos. Lima, Perú. Mc_Graw Hill.
- Jeff Fergurson, (Ed). (2003). La biblia de C++. USA. Anaya Multimedia.